Treball de Fi de Grau

Grau en Enginyeria Informàtica (GEI)

Performance Analysis and Optimization of an HPC application: DMRG++

Amb la col·laboració del
Barcelona Supercomputing Center (BSC-CNS)

1 de juliol de 2019

Autor: Joel Criado Ledesma

Directora: Marta Garcia Gasulla (BSC)

Codirector: Raül Sirvent Pardell (BSC)

Ponent: Jesús Labarta Mancho (Arquitectura de Computadors)

Especialitat: Enginyeria de Computadors

Quadrimestre: Primavera 2018-2019

Facultat d'Informàtica de Barcelona (FIB)

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Agraïments

Vull agrair a la Marta i el Raül, els meus directors, tota la dedicació que han posat en aquest projecte, per confiar en mi des que em vaig unir al seu equip, tot i les meves carències. Al Jesús, el meu ponent, per tots els seus consells i la seva gran ajuda per analitzar traces i dissenyar solucions; aquestes reunions acaben sent les millors sempre. Aquest treball l'he pogut acabar gràcies a ells, al cap i a la fi.

També, voldria agrair als meus companys de feina Aleix, Marc, Mario i Oleksandr per ajudar en tot el que han pogut, sobretot a riure i relaxar-me quan feia falta. Els cafès i esmorzars amb vosaltres són millors.

Per acabar, vull dedicar aquest treball al Quim i la Pilar, els meus pares. Sense vosaltres res d'això hauria arribat a passar mai. Gràcies pel vostre esforç i dedicació.

Resum

DMRG++ (Density Matrix Renormalization Group) és una aplicació de física de la matèria condensada orientada a HPC, originalment desenvolupada per l'Oak Ridge National Laboratory (ORNL). En aquest projecte es treballarà en la millora de la part de càlcul intensiu de l'aplicació, fent ús d'una *miniapp* que encapsula aquesta secció crítica. Partint d'una implementació inicial amb OpenMP basada en diversos *parallel for* aniuats, s'exploraran diferents alternatives per millorar el temps d'execució i el consum de memòria mitjançant el model de tasques amb dependències d'OpenMP, tot fent servir una estratègia d'anàlisi de l'aplicació i de desenvolupament iterativa. D'aquesta manera, no només esperem contribuir amb la millora d'una aplicació científica, sinó també mostrar tècniques d'anàlisi efectives i estratègies de paral·lelització per a aplicacions amb distribucions de feina molt desiguals.

Paraules clau: CAP | Optimització | Anàlisi | OpenMP | Tasques | Dependències

Resumen

DMRG++ (Density Matrix Renormalization Group) es una aplicación de física de la materia condensada orientada a HPC, originalmente desarrollada por el Oak Ridge National Laboratory (ORNL). En este proyecto se trabajará en la mejora de la parte de cálculo intensivo de la aplicación, haciendo uso de una *miniapp* que encapsula esta sección crítica. Partiendo de una implementación inicial con OpenMP basada en varios *parallel for* anidados, se explorarán diferentes alternativas para mejorar el tiempo de ejecución y el consumo de memoria mediante el modelo de tareas con dependencias de OpenMP, usando una estrategia de análisis de la aplicación y de desarrollo iterativa. De este modo, no solamente esperamos contribuir con la mejora de una aplicación científica, sino también mostrar técnicas de análisis efectivas y estrategias de paralelización enfocadas a aplicaciones con distribuciones de trabajo muy desiguales.

Palabras clave: CAP | Optimización | Análisis | OpenMP | Tareas | Dependencias

Abstract

DMRG++ (Density Matrix Renormalization Group) is a condensed matter physics application oriented to HPC, developed by Oak Ridge National Laboratory (ORNL). In this project, we will focus on improving the intensive arithmetic kernel of the application, using a *miniapp* that encapsulates this critical program part. Starting with an initial implementation with OpenMP, which uses several nested *parallel for*, we will explore different alternatives to improve its execution time and memory consumption through OpenMP task dependency model, taking advantage of an iterative strategy of in-depth application analysis and development. In this way, we are not just contributing by improving a scientific application, but also showing effective analysis techniques and best practices for programmability and parallelization focused on applications with irregular workloads.

Keywords: HPC | Optimization | Analysis | OpenMP | Tasks | Dependencies

Índex

Índex de figures

Índex de taules

Índex de Codis

Glossari

API: Application Programming Interface.

BSC-CNS: Barcelona Supercomputing Center - Centro Nacional de Supercomputación.

CAP: Computació d'Altes Prestacions.

CPU: Unitat Central de Processament / Central Processing Unit.

DIMM: Dual In-line Memory Module.

DMRG: Density Matrix Renormalization Group.

EDR: Endpoint Detection and Response.

FLOPS: Floating Point Operations Per Second.

FPGA: Field-Programmable Gate Array.

GCC: GNU Compiler Collection.

GEI: Grau en Enginyeria Informàtica.

GEP: Gestió del Projecte.

GPU: Unitat de Procés Gràfic / Graphic Processing Unit.

GNU: GNU Not Unix.

HPC: High Performance Computing.

HBM: High Bandwidth Memory.

IDE: Integrated Development Environment.

MPI: Message Passing Interface.

NUMA: Non-Uniform Memory Access.

ORNL: Oak Ridge National Laboratory.

POP: Performance Optimization and Productivity.

RAM: Random Access Memory.

SO: Sistema Operatiu.

TFG: Treball Final de Grau.

Capítol 1

Context

1.1 Introducció

Aquest projecte és un Treball Final de Grau (TFG) del Grau d'Enginyeria Informàtica (GEI) de la Facultat d'Informàtica de Barcelona (FIB), emmarcat dins de l'especialitat d'Enginyeria de Computadors. Ha estat desenvolupat en col·laboració amb el Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC-CNS) i l'Oak Ridge National Laboratory (ORNL), amb la finalitat de millorar el rendiment d'una aplicació, anomenada DMRG++, orientada a HPC (**H**igh **P**erformance **C**omputing) i en desenvolupament a l'ORNL.

DMRG++ és una aplicació Open Source per HPC escrita en C i C++ que resol un problema físic anomenat Density Matrix Renormalization Group, usat en sistemes de molts cossos. En aquest projecte hem treballat amb una mini aplicació (*miniapp*) que reprodueix la part de càlcul intensiu de l'original, per així poder estalviar recursos i temps a l'hora de fer les proves. El codi es troba disponible en [1].

Inicialment, l'aplicació presentava una estructura amb diversos bucles aniuats i dependències entre ells i una primera paral·lelització mitjançant OpenMP molt poc eficient, tant en temps d'execució com en memòria. El treball té com a objectiu reduir aquestes dues variables fent servir el mateix model de programació (OpenMP). Per aconseguir això serà indispensable seguir un mètode d'anàlisi d'aplicacions com el del projecte POP [3], l'ús d'eines d'anàlisis com Paraver i tenir sempre en compte el sistema hardware on es treballa.

1.2 Estat de l'art

Actualment, la Computació d'Altes Prestacions es dirigeix cap a l'*Exascale* (*i.e.*, sistemes capaços d'executar com a mínim un exaFLOP/s), i per assolir aquesta fita és necessari que tothom hi posi el seu esforç. Per això, són necessaris processadors, i components en general, més potents i eficients, models de programació més flexibles i escalables i aplicacions que puguin fer servir tot el potencial disponible per explotar el paral·lelisme.

Des del punt de vista de l'arquitectura, analitzant la llista del *top500* [5] podem observar que la majoria de sistemes es mouen cap a processadors amb més i més cores o bé inclouen diversos acceleradors (*e.g.*, GPUs). Pel que fa als models de programació i llibreries, els esforços es dirigeixen cap a implementacions i eines més flexibles [14, 20], que

ofereixen eines per paral·lelitzar programes amb distribucions de feina desiguals i dependències de dades, i permeten arreglar els desbalancejos de càrrega de treball en temps d'execució. Finalment, analitzant la direcció de les aplicacions científiques, podem observar esforços per desenvolupar-ne de noves i/o afegir noves funcionalitats, i per millorar el seu rendiment [9]; per això mateix, és necessari facilitar la tasca de programació, ja que les aplicacions no es poden reescriure cada cop que es canvia l'arquitectura, millorant els models de programació usats i els *runtimes* [15].

Si ens centrem en els models de programació actuals, trobem dos ben destacats avui en dia: MPI [22] (Message Passing Interface), dissenyat per a programació amb pas de missatges i memòria distribuïda, i OpenMP [24], pensat per a memòria compartida i el qual s'utilitzarà àmpliament durant aquest projecte.

OpenMP incorpora la directiva *task* des de la seva versió 3.0, que permet expressar concurrència amb blocs desestructurats i seccions irregulars, en contraposició a les directives *loop*, que tenen una estructura bastant més regular. Inicialment, les tasques eren independents entre elles, i el programador havia d'assegurar ell mateix la correctesa del programa. Amb la versió 4.0 es va incloure el model de tasques amb dependències, facilitant la feina dels programadors en gran mesura. Finalment, la versió 4.5 incloïa les tasques amb prioritats, que facilita la feina de planificació de les tasques. Però per molt que canviï l'especificació, els desenvolupadors han de modificar les aplicacions per treure tot el partit del potencial al seu abast. En aquest projecte mostrarem un cas pràctic d'aquesta situació, adaptant una aplicació amb patrons irregulars mitjançant el model de tasques amb dependències.

Schuchart et al. [28] avaluen diverses implementacions del model de tasques amb dependències i quin impacte té el nombre de dependències en el seu rendiment. En ell es mostra com les últimes versions dels compiladors de referència (i.e., Intel, Clang i GNU) mantenen un bon rendiment fins a 4~8 dependències (en funció del test en qüestió), punt en el qual el rendiment comença a decréixer bastant, així que és important mantenir el nombre de dependències baix. A més a més, s'hi pot observar l'*overhead* associat a la granularitat de les tasques (i.e., el seu temps d'execució), cosa que també es pot apreciar en l'estudi de Gautier et al. [16].

Pel que fa a altres estudis que es dediquin a optimitzar aplicacions de Computació d'Altes Prestacions amb OpenMP, trobem [21], on s'optimitza un programa mitjançant directives d'OpenMP de tipus *loop*, i també [18], on opten per aprofitar la programació heterogènia i usar les directives *target* d'OpenMP, pensada per usar dispositius com GPUs. Tot i que són un bon exemple de com utilitzar OpenMP a l'hora de millorar una aplicació, creiem que el nostre estudi pot ser interessant per qui vulgui fer-ho mitjançant tasques, ja que és un tema poc tractat en la literatura actual.

1.3 Motivació

Durant molt de temps, el disseny de processadors ha estat marcat per la Llei de Moore [23], doblant el nombre de transistors en un xip cada dos anys aproximadament. Això va permetre la millora del rendiment dels processadors a força d'incrementar la freqüència de funcionament, fins a arribar a un límit de potència consumida, a partir del qual les pèrdues per calor i corrents de fugues eren massa grans. Llavors, per donar utilitat al creixent nombre de transistors disponibles, els fabricants comencen a incloure més d'un *core*

(Element principal d'un processador, encarregat de realitzar les operacions principals. També anomenat CPU) per microprocessador. La Figura 1.1 mostra aquesta evolució en el disseny de processadors. Aquests processadors s'incorporen en els clústers ja existents, donant lloc als supercomputadors tal com els coneixem avui en dia. Amb l'aparició d'aquestes noves màquines la manera de programar i gestionar els recursos canvia, així que és necessari el sorgiment de sistemes operatius per suportar això i nous models de programació (e.g., MPI, OpenMP).

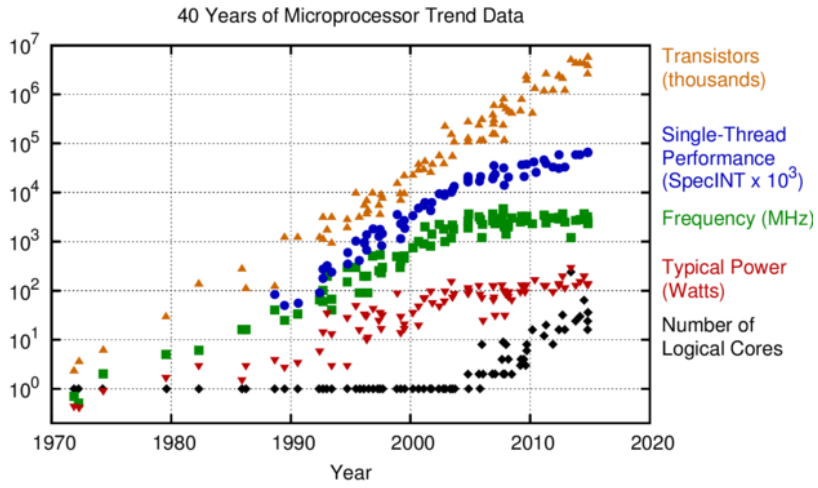

Figura 1.1: Evolució dels microprocessadors des del 1970 fins al 2015 [26]. Dades originals fins l'any 2010 recollides i dibuixades per M. Horowitz, F. Labonte, O. Shachman, K. Olukotun, L. Hammond, and C. Batten. Nou gràfic i dades pel període 2010-2015 per K. Rupp.

Moltes aplicacions que abans trigaven dies, setmanes o que ni tan sols s'havien plantejat comencen a ser viables gràcies al potencial que ofereixen els supercomputadors i és necessari programar-les adequadament perquè aprofitin els recursos disponibles de manera òptima. És en aquest context on apareix la Computació d'Altes Prestacions (HPC, per les seves sigles en anglès), especialitzada en optimitzar el màxim possible els programes per aprofitar els recursos disponibles de la millor manera. Actualment, l'HPC és usat àmpliament en molts camps científics, com per exemple en biomedicina, predicció climàtica o bé mecànica quàntica.

Un exemple d'aquestes aplicacions és DMRG++, en la qual se centra aquest treball. Com s'ha mencionat anteriorment, l'estat inicial d'aquest programa té un gran potencial de millora: les execucions actuals comporten un temps inassumible en moltes situacions, de l'ordre dels dies o setmanes. A causa d'això, es vol aplicar millores per a resoldre aquest inconvenient i permetre als científics obtenir els resultats desitjats en un temps inferior. Addicionalment, en reduir el temps d'execució i el consum de memòria també s'obre la porta a simular models físics més complexos (en dues o tres dimensions), cosa que ara mateix és impensable.

1.4 Estructura de la Memòria

La resta d'apartats de la memòria s'estructuren de la següent manera: Al Capítol 2 s'expliquen els objectius associats al projecte, els riscs que poden aparèixer durant el seu trans-

curs i com solucionar-los, la metodologia a emprar, i els actors implicats en la seva realització; el Capítol 3 introdueix el *hardware* i *software stack* usats en el projecte, així com el funcionament i estructura del codi de DMRG++; en el Capítol 4 es dona pas a l'estudi principal d'aquest treball, on es mostra la feina feta en cada versió del codi de DMRG++, comparant la nova versió amb l'anterior a cada pas, i s'inclou una comparativa final entre la versió original i l'última creada; el Capítol 5 descriu les tasques en les quals s'ha estructurat el projecte, estableix una previsió temporal i dependències entre elles i especifica els recursos necessaris per a dur-les a terme i el seu cost econòmic; el Capítol 6 avalua el projecte en els àmbits econòmic, ambiental i social, seguint la matriu de sostenibilitat explicada en el curs de GEP; i finalment el Capítol 7 explica les conclusions del projecte.

Capítol 2

Abast del projecte

En aquest capítol es tractarà l'abast del projecte, els seus objectius i la manera d'assolir-los. Tot això inclou la motivació darrere del projecte, la metodologia de treball emprada, els requeriments i els possibles riscos que cal assumir.

2.1 Requerimients

- Reduir el temps d'execució total de l'aplicació de l'estudi.
- Reduir el consum de memòria de l'aplicació.
- Programar tenint en compte l'arquitectura Power9 d'IBM, usant les llibreries específiques per aquest sistema, donat que és l'arquitectura de la màquina disponible a l'ORNL.
- Emprar bones pràctiques de programació, amb un bon estil i afegint la mínima complexitat possible al codi.
- Mantenir un contacte constant i fluid, tant amb l'equip del BSC com amb el de l'ORNL.
- Familiaritzar-se amb el clúster experimental CTE-Power del BSC i als programes disponibles en ell.

2.2 Riscs i possibles solucions

A continuació, es defineixen els riscs que poden ocórrer durant el desenvolupament del projecte, així com les possibles solucions a aquests.

No disponibilitat del CTE-Power

Totes les proves es realitzen al CTE-Power (explicat a la Secció 3.1) del BSC, així que en cas de manteniment o caiguda de la màquina és impossible provar qualsevol canvi.
Solució: En aquests casos no es podrà executar cap prova ni intentar extreure resultats de temps/memòria, però es podrà seguir treballant amb el codi a qualsevol altra màquina o bé dedicar aquest temps a escriure documentació.

Augment del temps de les tasques

És possible que en algun punt del projecte el desenvolupador principal s'endarrereixi en completar les tasques segons la planificació. Hi poden haver diversos motius: mala planificació, malalties, baixes laborals, tasques més complexes del que s'esperava, etc.
Solució: Cal elaborar la planificació amb cura, preveure que aquestes situacions es poden donar, i deixar marges suficients per resoldre qualsevol problema.

Fallada de l'equip

Es pot donar el cas de fallada de l'equip de treball usat pel desenvolupador, o bé algun element com el monitor o el teclat.
Solució: Com es disposa d'un portàtil, es pot seguir treballant mentre es proveeix d'un recanvi, sempre que l'element que ha fallat no sigui el mateix portàtil. En aquest cas es perdrien un mínim de dos dies mentre el departament de suport del BSC el revisa i prepara un de nou o bé retorna l'anterior en funcionament. No s'han de guardar dades importants només en local, cal fer servir sempre el sistema d'arxius del BSC, que compta amb còpies de seguretat periòdiques, i/o algun sistema de control de versions.

Temps d'espera d'execució

Treballar amb un clúster implica compartir un recurs amb molts més usuaris i usar cues d'execució per gestionar-lo. Per tant, és possible que en algun moment no es pugui executar el programa a l'instant i haver d'esperar algunes hores.
Solució: Tot i que la situació és freqüent en moltes de les màquines del centre, les espere en la màquina CTE-Power no solen ser molt llargues. Aquest temps es pot dedicar a seguir modificant el codi o a escriure documentació.

Diferència horària amb els Estats Units

L'equip d'ORNL es troba a Knoxville, Tennessee, als Estats Units, fet que crea una diferència horària de sis hores amb Barcelona. Això dificulta el fet de poder establir conferències entre els dos equips i l'enviament de missatges per correu electrònic, que sol ser poc constant per culpa d'això. El fet que a l'equip del BSC acostumem a treballar des de primera hora del matí, quan allà és de nit, accentua el problema.
Solució: Difícilment es pot canviar la diferència horària, però sí que es pot estar pendent al correu electrònic durant tot el dia i treballar alguna hora a la tarda quan sigui necessari assistir a una videoconferència.

2.3 Metodologia

En aquesta secció es donen a conèixer els mètodes de treball fets servir en el projecte, les eines de seguiment i els mètodes de validació i avaluació.

2.3.1 Mètodes de treball

Durant tot el projecte es seguirà una forma de treball iterativa. És a dir, s'analitzarà el codi per veure què es pot millorar i com, s'aplicaran aquestes millores, es validarà el comportament de l'aplicació i es mesuraran les mètriques de rendiment rellevants. Un cop s'arriba a aquest punt, el procés es torna a repetir fins a assolir el resultat objectiu. Al final de

cada iteració, serà necessari documentar adequadament tots els canvis realitzats al codi i les anàlisis pertinents per poder tenir un històric del treball realitzat, explicar la feina feta als desenvolupadors d'ORNL i poder escriure un informe final en acabar el projecte. Amb aquest mètode es busca reduir la complexitat de cada tasca individual, facilitant el treball i agilitzant-lo.

A causa d'això, la metodologia SCRUM [25] és la que millor s'adapta a la nostra intenció de treball. Ofereix un mètode de treball incremental, on la planificació es pot anar modificant a mesura que avança el projecte, si és necessari, i el producte es pot provar constantment. A més a més, ofereix una major flexibilitat a l'hora de treballar, permetent superposar tasques en alguns casos i ajudant a reduir els riscs del projecte.

2.3.2 Eines de seguiment

Els clústers del BSC són un entorn bastant tancat, cosa que dificulta l'ús d'una eina de control de versions pel codi. Es treballarà essencialment amb el sistema de fitxers del BSC, el qual és accessible des de la xarxa interna sense cap problema i se'n fan còpies de seguretat periòdicament, així que difícilment es perdran dades. Addicionalment, es disposarà d'un repositori de GitHub [17] compartit entre el BSC i l'ORNL per intercanviar fitxers entre equips i mantenir un control de la feina feta per les dues bandes.

També es farà servir l'eina *online* Trello [6] per a tenir un control de la feina feta pel desenvolupador i proposar-ne de nova. Es tracta d'una eina creada per facilitar l'organització de projectes, on es creen les tasques a realitzar i es van movent entre diferents taulells, permetent seguir en tot moment l'estat de la tasca en qüestió. S'adapta perfectament a la metodologia triada pel projecte (SCRUM).

A més a més, es realitzaran reunions periòdiques entre l'equip del BSC i esporàdiques amb l'equip de l'ORNL per validar el progrés realitzat. En aquestes reunions es pot debatre la correctesa del codi, suggerir canvis a realitzar i aprofundir en l'anàlisi de l'aplicació. Són una eina excel·lent perquè tothom pugui aportar al projecte i ajudar al desenvolupador principal a observar coses que pot estar passant per alt.

2.3.3 Mètodes de validació

Com ja s'ha indicat a la Secció 2.3.1, serà necessari verificar que el programa funciona correctament a cada iteració del desenvolupament. Per a realitzar aquesta tasca és necessari comparar els resultats amb els obtinguts amb el codi inicial, que s'utilitzarà sempre com a referència. Un cop s'obté una versió amb la qual s'està satisfet, s'envia el codi a ORNL fent servir el repositori compartit de GitHub perquè el puguin verificar amb inputs més realistes (*e.g.*, de major tamany), ja que la mini aplicació amb la qual es treballa no cobreix tot l'espectre de proves del programa real.

Per tal de validar el rendiment de l'aplicació es faran servir Paraver i Extrae (Seccions 3.3.1 i 3.3.2, respectivament). També s'usaran eines per obtenir temps d'execució i el consum de memòria, les dues mètriques clau en el projecte. Per acabar, pot ser necessari l'ús de *profilers* com Valgrind [7] per obtenir detalladament l'evolució de la memòria durant tota l'execució del programa i poder validar el seu bon funcionament.

Finalment, les reunions periòdiques també serviran per anar validant els resultats obtinguts. Els directors del projecte i altres companys amb més experiència podran escoltar la

feina feta pel desenvolupador i podran suggerir canvis a realitzar, indicar que algun pas s'ha fet de manera incorrecta o assenyalar que el projecte va per bon camí.

2.3.4 Avaluació del resultat final

Un cop s'obtingui un resultat parcial amb el qual s'està satisfet i es cregui que el temps necessari per millorar el codi sigui excessiu, es pot donar el resultat com a definitiu. Fixar un punt exacte des de l'inici pot ser contraproduent, ja que es pot tirar tant a l'alça com a la baixa i errar el resultat per bastant. A mesura que el projecte progressi i es detecti que el temps utilitzat en aplicar qualsevol millora no compensa el guany obtingut vol dir que és moment d'aturar-se. És a dir, dedicar una setmana per reduir el temps d'execució un 1,01× no és rentable, i aquesta situació és un indicatiu per acabar amb la fase d'optimitzacions.

Per acabar el projecte, serà necessari avaluar el rendiment i consum de memòria de la mateixa manera que en els apartats anteriors, fer una comparativa de l'evolució del programa al llarg de tot el projecte i presentar un document amb aquestes dades i explicant tots els canvis realitzats.

2.4 Actors Implicats

A continuació es detallaran totes les persones que col·laboren i/o han col·laborat en el projecte, així com aquelles que se'n poden arribar a beneficiar, directe o indirectament.

- **Desenvolupador**: És l'encarregat de dur a terme el projecte. Això inclou principalment la feina d'anàlisi i millora de l'aplicació i verificació d'aquestes. També serà el responsable d'elaborar qualsevol documentació necessària i de cercar informació.
- **Directora, Codirector i Ponent**: S'encarreguen de supervisar el projecte i aconsellar al desenvolupador amb l'objectiu de complir les fites marcades. Organitzen reunions per verificar el progrés del projecte quan és necessari, ja sigui entre l'equip del BSC-CNS o amb la gent d'ORNL. Finalment, també donen suport en la tasca d'anàlisis de l'aplicació quan és necessari.
- **Barcelona Supercomputing Center**: El centre de treball és l'encarregat d'aportar el material usat pel projecte: portàtil, monitor, teclat, fungibles... A més a més, també posa a disposició tots els clústers disponibles, i en especial el CTE-Power, sense els quals el projecte seria impossible.
- **Oak Ridge National Laboratory**: S'encarreguen d'adaptar les millores, implementades pel desenvolupador en la *miniapp*, a l'aplicació real. Organitzen reunions amb l'equip del BSC quan ho creuen necessari, resolen dubtes sobre el funcionament del programa real i col·laboren a decidir els següents passos en la investigació.
- **Beneficiaris**: Els principals beneficiaris seran els desenvolupadors i científics d'ORNL, que podran executar el programa en menys temps i consumint menys memòria, fet que permet augmentar la mida de l'input usat. Indirectament, qualsevol científic extern interessat a fer servir el programa es veurà beneficiat de la mateixa manera. Finalment, qualsevol altre desenvolupador d'aplicacions HPC amb patrons de paral·lelització pot aplicar les tècniques emprades en aquest projecte, aprofitant el treball que ja s'ha realitzat, ja que es donen les directrius necessàries per poder millorar el rendiment.

Capítol 3

Entorn

En aquest capítol es dona a conèixer l'entorn usat al llarg del projecte. Primer de tot, es detalla l'arquitectura i el *software stack* principal que s'ha fet servir. Seguidament, es veuen els models de programació de memòria compartida que s'han considerat en aquest projecte i les eines d'anàlisi principal fetes servir. Per tancar aquest apartat, s'explica en què consisteix l'aplicació central del projecte i com s'estructura el seu codi.

3.1 Clúster CTE-Power

El CTE-Power [4] és un clúster propietat del BSC-CNS basat en l'arquitectura Power9 [27] d'IBM. Està compost per 2 nodes de login i 52 de càlcul, tots ells amb 2 IBM Power9 8335-GTH 2.4 GHz, 512 GB de memòria distribuïts en 16 DIMMs i 4 GPUs NVIDIA V100 amb 16 GB de memòria HBM2 (tot i que no es fan servir en aquest estudi). Com a SO s'utilitza Red Hat Enterprise Linux Server 7.5 i es fa servir una xarxa d'interconnexió Mellanox EDR.

A l'hora de compilar els codis hem fet servir sempre GCC 8.1.0, la versió més recent instal·lada a l'inici del projecte. També estan disponibles el compilador d'IBM (xlc) i el de PGC (pgcc), però vam preferir l'opció de GCC, ja que està disponible en altres màquines Linux i això ens permetria comparar rendiments mantenint el compilador estable, en cas que ho desitgéssim. Finalment, hem fet servir CMake 3.11.1 per generar els Makefiles (fitxer que conté un seguit de directives per compilar un programa i generar l'executable) i hem linkat els executables amb la llibreria científica d'IBM ESSL 5.4, que s'encarrega de realitzar les operacions matemàtiques explicades a la Secció 3.4.

3.2 Models de Programació Paral·lela

A l'hora d'implementar les millores de paral·lelisme s'ha debatut entre utilitzar OpenMP o OmpSs. S'ha decidit utilitzar el primer perquè és un estàndard pel que fa a paral·lelisme de memòria compartida i la majoria de compiladors el suporten. A més a més, les característiques que creiem més necessàries per al projecte estan incloses en l'estàndard OpenMP (i.e., tasques i dependències). A continuació s'expliquen breument els dos models.

3.2.1 OpenMP

OpenMP és una interfície de programació d'aplicacions (API) pensada per multiprogramació amb memòria compartida. Actualment és suportada per una gran diversitat de

compiladors i en moltes arquitectures diferents, sent un dels models de programació més estesos. L'ús d'OpenMP consisteix en anotar el codi amb certes directives per expressar el paral·lelisme i que després el compilador tradueixi en el codi adient; a més a més, l'API també inclou un seguit de rutines i variables d'entorn.

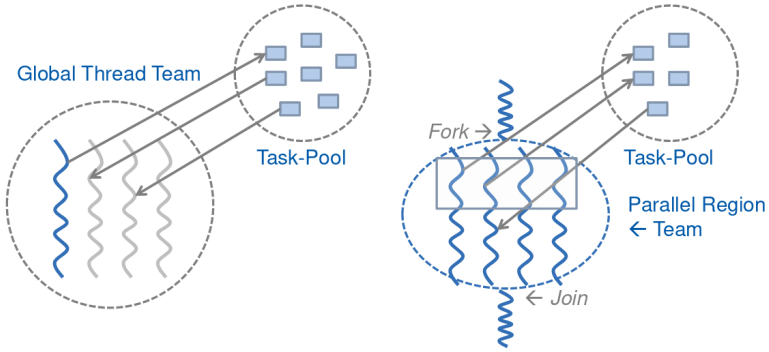

Figura 3.1: Comparació Thread-Pool i Fork-Join [29]

OpenMP utilitza el model *fork-join*, explicitant unes regions paral·leles amb una directiva. A l'inici d'aquestes la tasca genera diferents fils d'execució (*fork*) i en acabar tota la feina s'unifiquen tots els fils creats (*join*). Es pot veure un esquema a la dreta de la Figura 3.1.

Existeixen diverses implementacions d'OpenMP des de l'any 1997 fins a l'actualitat, amb la versió 5.0 de Novembre de 2018. En aquest projecte usarem en tot moment la versió 4.5 [24], ja que en el moment d'inici cap compilador implementava els nous canvis de la versió 5.0 (a causa del seu curt temps de vida) i que aquesta ja ofereix tot el seguit d'eines que necessitem (e.g., tasques, *taskloops*, dependències).

A continuació s'expliquen els diversos termes relacionats amb OpenMP que aniran apareixent al llarg del projecte:

- *Thread* OpenMP: Un *thread* és una entitat en execució amb una pila i una memòria estàtica associada. Un *thread* d'OpenMP és qualsevol *thread* manejat pel *runtime* d'OpenMP.

- *Parallel*: Directiva d'OpenMP per expressar el paral·lelisme. Aquesta directiva marca l'inici i el final d'una regió paral·lela (*fork-join* vist a dalt). Cal destacar que al final hi ha un punt de sincronització (barrera) per tots els *threads*, cosa que tindrà un pes important en les primeres versions d'aquest projecte.

- *For*: Directiva per especificar que les iteracions del següent bucle s'executaran en paral·lel. Les iteracions es reparteixen entre els diferents *threads*, de tal manera que no hi hagi més d'un executant una mateixa iteració.

- *Single*: Directiva per indicar que el següent bloc de codi només l'ha d'executar un dels *threads*. La resta de *threads* esperaran al final del bloc que aquest acabi, si no s'ha especificat el contrari.

- *Task*: Directiva que explicita la creació d'una tasca. En OpenMP, una tasca és una instància específica de codi executable i el seu entorn (dades).

- *Dependència*: Modificador de la directiva *task*. Indica que l'execució d'aquella tasca depèn d'una altra, o bé que alguna tasca depèn de la que té la directiva. Per establir aquesta dependència, s'utilitzen regions de memòria. En la versió usada d'OpenMP (4.5) n'existeixen tres tipus:
  - *In*: Si algun dels elements coincideix amb el d'alguna tasca prèviament generada amb una dependència *out* o *inout*, llavors la tasca amb *in* no s'executa fins que les altres acabin.
  - *Out*: Si algun dels elements coincideix amb el d'alguna tasca prèviament generada amb una dependència *in*, *out* o *inout*, llavors la tasca amb *out* no s'executa fins que les altres acabin.
  - *Inout*: Si algun dels elements coincideix amb el d'alguna tasca prèviament generada amb una dependència *in*, *out* o *inout*, llavors la tasca amb *inout* no s'executa fins que les altres acabin.
- *Prioritat*: Modificador de la directiva *task*. S'utilitza per donar indicis de quines tasques són més prioritaries, és a dir, que és més adient executar-les primer.
- *Barrera*: Punt de sincronització en el qual tots els *threads* aturen la seva execució (*i.e.*, deixen de fer feina útil) fins que tots arriben a aquest punt. Hi ha una directiva que explicita aquests punts (*barrier*), tot i que també existeixen directives que en tenen d'implícits en el seu final (*e.g.*, *parallel*, *single*)
- *Condició de carrera*: Situació que es dona quan dos o més *threads* intenten accedir a un mateix recurs (dada), que acaba desembocant en un comportament indeterminat, i sovint erroni, del programa.
- *Granularitat*: Terme que indica quanta feina té associada una tasca.

3.2.2 OmpSs

OmpSs [12] és un model de programació de memòria compartida desenvolupat pel BSC. Està basat en un antic model anomenat StarSs i té com a objectiu desenvolupar noves extensions per a OpenMP. A causa d'això, funciona amb diferents arquitectures, fins i tot GPUs i FPGAs, i amb Fortran, C i C++. Com l'objectiu principal és estendre OpenMP, el seu ús és molt similar, anotant el codi amb directives; de fet, un mateix codi pot arribar a funcionar amb els dos models sense cap problema. Per fer servir OmpSs és necessari l'ús del compilador Mercurium i del runtime Nanos++.

El model d'execució d'OmpSs està basat en tasques i dependències entre elles, amb un *Thread-Pool* per executar-les. A l'inici de l'execució, el procés genera un conjunt de fils d'execució, sense cap feina a realitzar; en el moment en el qual es genera una tasca aquesta és assignada a un fil d'execució, si n'hi ha de disponibles, o bé s'afegeix a una cua de tasques pendents. Es pot veure un exemple a l'esquerra de la Figura 3.1.

3.3 Eines d'Anàlisi

Per a realitzar la tasca d'anàlisi ens hem decantat pel conjunt de Paraver i Extrae, ambdues desenvolupades pel BSC. Això fa que el contacte amb l'equip de desenvolupament sigui molt més fàcil i fluid, resolent qualsevol dubte i/o problema que sorgeixi en poc temps.

També, el fet que siguin del BSC fa que estiguin instal·lades a totes les màquines i no ens faci falta preocupar-nos de realitzar instal·lacions ni actualitzacions. Per últim, tot l'equip del BSC ja tenim experiència prèvia en el seu ús, així que no serà necessari dedicar temps addicional a aprendre a fer servir cap eina nova. A continuació podeu trobar una breu descripció de les dues eines.

3.3.1 Paraver

Paraver [2] és una eina d'anàlisis *post-mortem* que permet visualitzar el comportament que ha tingut una aplicació durant la seva execució. Ve amb una gran varietat de fitxers de configuració, que permeten observar diferents esdeveniments, com per exemple l'estat de cada *thread* en un moment donat o els cicles per segon. Funciona amb els models de programació més habituals (e.g., MPI, OpenMP, Java, CUDA) i amb els propis del BSC, té suport per la llibreria de comptadors hardware PAPI i funciona en els sistemes operatius més populars.

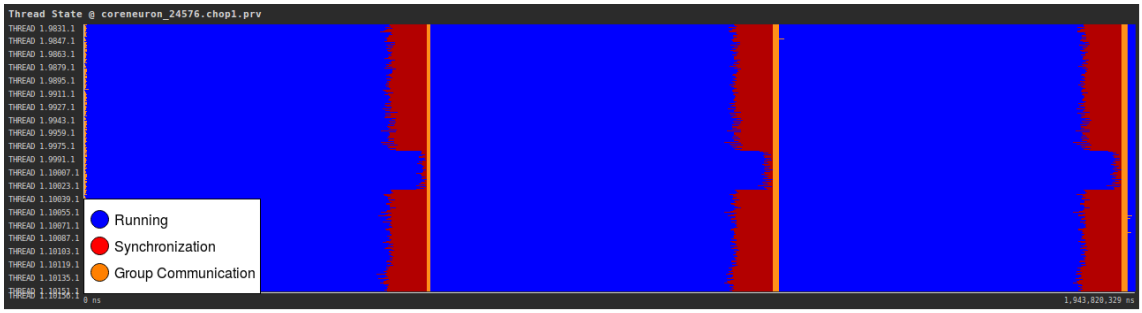

Figura 3.2: Cronograma d'exemple

La Figura 3.2 mostra un cronograma (també anomenat *traça*) d'exemple. Tot i que Paraver pot mostrar més coses a banda dels cronogrames, com ara histogrames i taules, l'ús que se'n fa en aquest treball és principalment visualitzar traces. En totes elles, el programa ens dibuixa els diferents *threads* fets servir en l'eix y, mentre que en l'eix x es representa el temps. En aquest cas, es mostren els estats pels quals han passat els *threads* durant un tros d'una execució, però com s'ha dit, existeixen moltes configuracions diferents per mostrar altres esdeveniments.

En aquesta traça es pot observar el que es coneix com a desbalanceig de càrrega. És un fet que es dona quan els processos no tenen el mateix volum de feina i, en arribar a una barrera, uns han d'esperar que arribin els que tenen més feina. En la Figura 3.2 podem veure com els *threads* del centre estan més estona en l'estat d'execució (*running*), incrementant el temps que passen esperant-se la resta (*synchronization*).

3.3.2 Extrae

Extrae [19] és una llibreria que facilita la captació d'esdeveniments (*e.g.*, event de PAPI, creació d'una tasca OpenMP, canvi d'estat d'un thread) en un codi, per tal de posteriorment generar traces de Paraver i fer-ne una anàlisi *post-mortem*. En executar el programa amb Extrae, aquest s'interposa en les rutines de l'aplicació i certs punts en concret per a reunir mètriques de rendiment d'interès per a l'usuari. Addicionalment, ofereix una API pròpia perquè l'usuari elabori els seus propis esdeveniments i pugui limitar l'ús d'Extrae a certes regions del codi.

3.4 DMRG++

Density Matrix Renormalization Group [30] és un mètode numèric que es fa servir en física quàntica per obtenir les propietats físiques dels sistemes de molts cossos. En aquest cas, s'usa per estudiar les propietats superconductores dels materials. DMRG++ [8] és una aplicació *open source*, escrita en C i C++, que implementa aquest mètode numèric. El programa simula models d'Hubbard, usats per descriure les transicions de sistemes conductors a aïllants, fent servir àlgebra de matrius disperses. La primera versió data de l'any 2009 i actualment segueix en desenvolupament.

A l'hora de preparar l'aplicació pels canvis cap a arquitectures *Exascale*, es va desenvolupar una mini aplicació [10, 13] que encapsula la part crítica en l'àmbit algorítmic i computacional (*i.e.*, Productes de Kronecker). Aquesta *miniapp* ens permet executar la part rellevant pel nostre estudi en molt menys temps, reduint hores de treball i consum energètic. Millorant el rendiment i el consum de memòria, estem contribuint a la seva portabilitat per les noves arquitectures *Exascale*, que permetran als científics no només obtenir els resultats en menys temps, sinó també treballar amb sistemes de més dimensions.

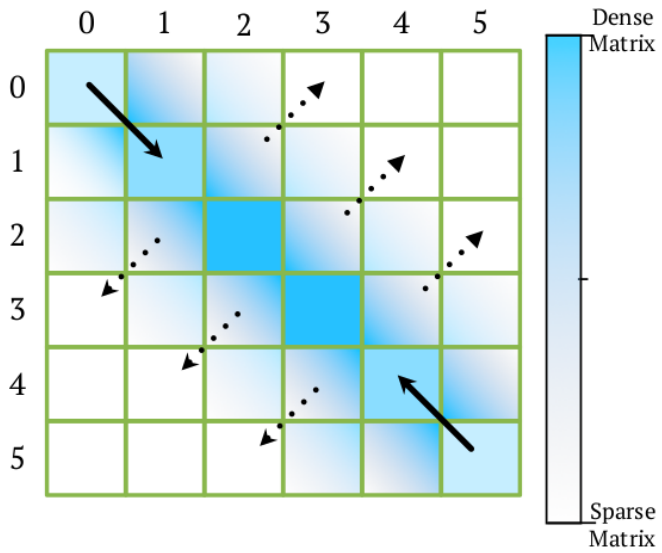

Figura 3.3: Distribució de la feina al llarg de la Matriu Hamiltoniana [10].

A la Figura 3.3 podem veure com es distribueix la feina en una Matriu Hamiltoniana, element que es fa servir en l'aplicació per emmagatzemar les dades. Així doncs, els elements del centre de la matriu, sobre la diagonal principal, són més densos (*i.e.*, comporten un volum de feina més elevat), mentre que els més allunyats representen dades molt disperses. Aquesta distribució de feina implica que el volum de treball varia en gran mesura en funció del punt de l'execució on es trobi el programa, com veurem en futures seccions.

A l'esquerra de la Figura 3.4 es mostra una representació simplificada de les estructures de dades usades en l'aplicació, així com l'operació entre les matrius i un vector, denominat Producte de Kronecker. En aquest cas, es representa una Matriu Hamiltoniana 2×2, on els elements dels diferents vectors A i B són vectors de mida variable, cosa que incrementa el desbalanceig que un pot pensar que existeix inicialment. D'altra banda, a la dreta de la Figura 3.4 es mostra el temps d'execució (eix y) de les diferents iteracions (eix x) de

DMRG++. Com es pot observar, aquest va creixent a mesura que l'execució avança, fruit dels canvis de fase, que comporten un increment de la grandària de la matriu.

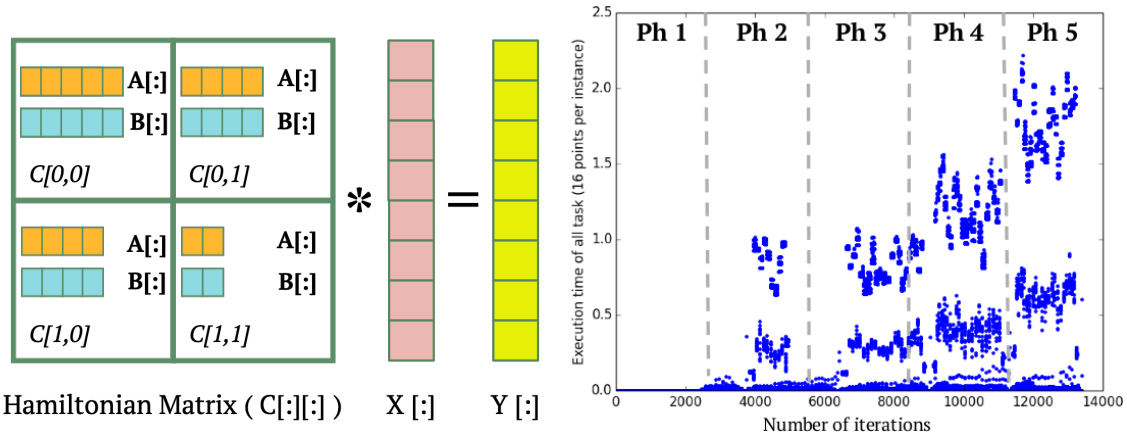

Figura 3.4: Estructures de dades i esquema de les operacions de DMRG++ [13] (esquerra). Distribució de la feina al llarg d'una execució de DMRG++ [10] (dreta).

3.4.1 Estructura de la mini aplicació

La mini aplicació de DMRG++ se centra en la part de càlcul intensiu, que realitza els Productes de Kronecker. Consta d'unes 1200 línies de codi en C i C++, i està paral·lelitzada amb OpenMP. La *miniapp* ve amb tres *inputs* diferents, cada un d'ells associat a una mida de problema de l'aplicació real (Fases 1, 3 i 5 de la Figura 3.4). La paral·lelització original es pot veure al Codi 3.1.

```
#pragma omp parallel for schedule(dynamic,1)
for(int ipatch=0; ipatch<npatches; ipatch++){
  std::vector<double> YI(vsize[ipatch], 0.0);
  #pragma omp parallel for schedule(dynamic,1) reduction(vec_add:YI)
  for(int jpatch=0; jpatch<npatches; jpatch++){
    std::vector<double> YIJ(vsize[ipatch], 0.0);
    #pragma omp parallel for schedule(dynamic,1) reduction(vec_add:YI)
    for(int k=0; k<CIJ.cij[ipatch][jpatch].size()){
      std::vector<double> Y_tmp(vsize[ipatch], 0.0);
      Matrix A = CIJ.cij[ipatch][jpatch]->A[k];
      Matrix B = CIJ.cij[ipatch][jpatch]->B[k];
      int has_work = (A->nnz() && B->nnz());
      if(!has_work) continue;
      A->kron_mult('n','n', *A, *B, &X[j1], &Y_tmp[0]);
      for(int i=0; i<vsize[ipatch]; i++)
        YIJ[i] += Y_tmp[i];
    }
    for(int i=0; i<vsize[ipatch]; i++)
      YI[i] += YIJ[i];
  }
  for(int i=i1; i<i2; i++)
    Y[i] = YI[i-i1];
}
```

Codi 3.1: Paral·lelització original

Com es pot observar, el codi consta de tres bucles *for* (*ipatch*, *jpatch*, *k*) on es realitzen crides a *kron_mult* (línia 14), funció encarregada de realitzar els Productes de Kronecker esmentats. Aquesta funció rep com a paràmetres d'entrada dues 'n', que indiquen que les matrius són normals i no transposades, les matrius A i B i el vector X, amb els quals es realitzen les operacions. Per acabar, també rep la direcció d'un vector *Y_tmp*, que serveix per retornar el resultat de les operacions.

També es pot veure l'ús d'estructures auxiliars per acumular resultats parcials i evitar condicions de carrera (línies 3, 6 i 9), que finalment s'acaben sumant al vector Y (línies 16, 19 i 22), que és el paràmetre de retorn. El codi inclou tres directives *for* d'OpenMP en els bucles principals (línies 1, 4 i 7), amb una reducció sobre els vectors *YI* i *YIJ* en els bucles *jpatch* i *k*, respectivament. El fet d'optar per aquesta estratègia genera un consum de memòria addicional a causa dels vectors de reduccions i no permet aprofitar el paral·lelisme de forma òptima, a causa del gran desbalanceig que hi ha entre els diferents *parallels* i les barreres implícites que tenen al final. En la Secció 4.1 s'entrarà més en detall en aquest tema.

Capítol 4

Anàlisi i Optimitzacions

En aquest capítol s'expliquen tots els passos seguits per millorar la *miniapp* presentada anteriorment. Per cada versió (excepte l'inicial, que ja s'ha introduït a la Secció 3.4) s'inclou la porció de codi rellevant, s'expliquen les modificacions realitzades i s'analitza el rendiment d'aquesta versió amb l'ajut de traces de Paraver. El procés de millores ha estat iteratiu i incremental, així que cada versió que es mostra parteix de l'anterior i per aquesta raó també la comparem amb l'anterior. Pel que fa al consum de memòria, es comenta per totes les versions juntes en la Secció 4.7.

Tots els resultats mostrats són mitjanes de 5 execucions independents de 10 iteracions consecutives; com que l'error relatiu està sempre per sota del 5% no es mostren les barres d'error. A l'hora de realitzar els experiments d'escalabilitat, s'han prés les mesures per 1, 2, 4, 8, 16, 20 (coincideix amb el nombre de *cores* per *socket*) i 40 (nombre total de *cores* físics disponibles en un node) *threads*. Totes les traces de Paraver mostrades en aquest estudi han estat obtingudes amb la versió 3.5.4 d'Extrae, la més recent en el moment d'iniciar el projecte. Totes les execucions s'han realitzat en un sol node del CTE-Power en exclusivitat d'execució (cap més usuari treballant a la vegada).

4.1 Anàlisi inicial

En la Figura 4.1 podem observar una traça de l'execució de la *miniapp* amb el segon *parallel for* activat (*jpatch*). A l'eix x hi ha representat el temps, mentre que a l'eix y es representen els diferents *threads* OpenMP, 40 en aquest cas. Pel que fa al color, aquest indica la duració de l'esdeveniment, en aquest cas la durada de les ràfegues de computació útil; el gradient va de verd a blau, amb el verd indicant valors petits i el blau grans, mentre que les zones blanques representen temps en el qual els processadors estaven sense feina (*idle*). A la part inferior podem veure el número de *threads* actius com a funció, amb valors entre 0 i 40, tant en aquesta figura com en les següents traces de Paraver.

En aquesta traça es pot observar com la feina dels diferents *parallel loops* no és homogènia al llarg de l'execució. No només hi ha desbalanceig de la càrrega de treball dins la mateixa iteració del bucle (representat per les columnes), sinó que cada iteració té un volum de feina diferent. Tot això ens indica que el principal coll d'ampolla de l'aplicació és el balanceig de la càrrega, encara que molta d'aquesta feina es podria fer perfectament en paral·lel. Per solucionar el problema farem servir el model de tasques d'OpenMP, que ens ajudarà a exposar el potencial paral·lelisme de l'aplicació, tot evitant les barreres periòdiques que comporten els *parallels*.

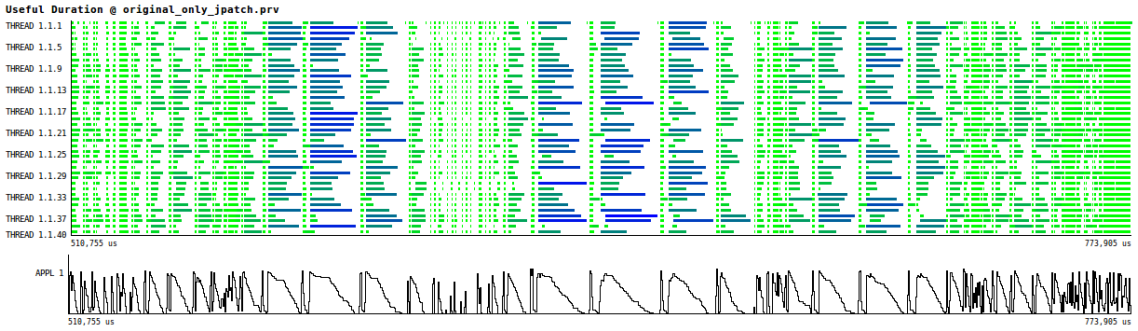

Figura 4.1: Cronograma de la versió Original de la duració de la feina útil

4.2 Primera Tasquificació

Com s'explica a la Secció 4.1, el motiu principal que limita el rendiment és el *load balance*, cosa que empitjora per culpa de les barreres implícites que hi ha al final de cada *parallel*. Com a conseqüència, el primer canvi ha consistit a eliminar els *parallels* aniuats (i, en conseqüència, les barreres) i fer servir tasques en el seu lloc. En el Codi 4.1 podem veure la porció de codi corresponent.

```
1  #pragma omp parallel
2  #pragma omp single
3  for(int ipatch=0; ipatch<npatches; ipatch++){
4    for(int jpatch=0; jpatch<npatches; jpatch++){
5      for(int k=0; k<CIJ.cij[ipatch][jpatch].size(); k++){
6        double* Y_tmp = new double[vsize[ipatch]]();
7        Matrix A = CIJ.cij[ipatch][jpatch]->A[k];
8        Matrix B = CIJ.cij[ipatch][jpatch]->B[k];
9        int has_work = (A->nnz() && B->nnz());
10       if(has_work){
11         //Tasca encarregada dels productes de kronecker.
12         #pragma omp task depend(inout:Y_tmp[0:vsize[ipatch]]) \
13         firstprivate(A,B)
14         A->kron_mult('n','n', *A, *B, &X[j1], &Y_tmp[0]);
15         //Tasca de reduccio.
16         #pragma omp task depend(inout: Y[i1:i2]) \
17         depend(in: Y_tmp[0:vsize[ipatch]])
18         {
19         int ilocal=0;
20         for(int i=i1; i<i2; i++) Y[i] += Y_tmp[ilocal++];
21         delete[] Y_tmp;
22         }
23       }
24     }
25   }
26 }
```

Codi 4.1: Primera Tasquificació

Es manté una sola regió paral·lela (línia 1) i afegim una regió single (línia 2) per crear les tasques. Aquí trobem dos tipus de tasques: de computació (línia 12), encarregades de realitzar la part de càlcul intensiu (Productes de Kronecker), i de reducció (línia 16), encarregades d'acumular els resultats parcials de la primera tasca en el vector de retorn. A

l'hora de passar els resultats d'una tasca a l'altra, es fan servir vectors temporals (línia 6), que s'encarrega de crear el *thread* principal seqüencialment. Per poder garantir que el resultat del programa és l'adequat, la primera tasca té una dependència de tipus *inout* sobre el vector temporal, mentre que la segona tasca la té de tipus *in*, garantint que s'executen en l'ordre correcte. Finalment, les tasques de reducció també tenen una dependència de tipus *inout* sobre una porció del vector Y (vector de retorn) amb la intenció d'evitar que més d'una tasca escrigui sobre la mateixa posició del vector a la vegada.

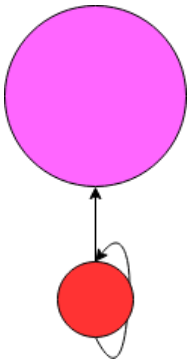

Figura 4.2: Dependències entre tasques: Primera Tasquificació

En la Figura 4.2 es mostren els dos tipus de tasques i les dependències entre elles, de tal manera que el color *rosa* està associat a les tasques de càlcul i el *vermell* a les tasques de reducció . També, es pot veure com les tasques *vermelles* depenen d'una altra tasca *vermella* i d'una *rosa*. Finalment, és important veure que els dos tipus de tasca es representen amb un tamany diferent, indicant que les de color rosa tenen una granularitat major.[1]

La Figura 4.3 mostra una traça de Paraver d'aquesta versió, on els colors coincideixen amb la Figura 4.2. Com es pot apreciar, no s'observa cap element de sincronització, la qual cosa permet executar diverses tasques rosa concurrentment i explotar millor el paral·lelisme. Tot i això, aquesta versió presenta alguns problemes: a) Un sol *thread* s'encarrega de la creació de tots els *buffers* temporals, cosa que afegeix un *overhead* important i incrementa el consum de memòria, i b) la duració de les tasques de càlcul presenta una gran variabilitat, amb tasques que triguen uns 18 μs, mentre que la mitja està en 150 μs. A la part inferior de la figura es pot veure la influència que té l'*overhead* de la creació de tasques, amb zones on gairebé no hi ha tasques executant-se, ja que la producció no pot seguir el ritme.

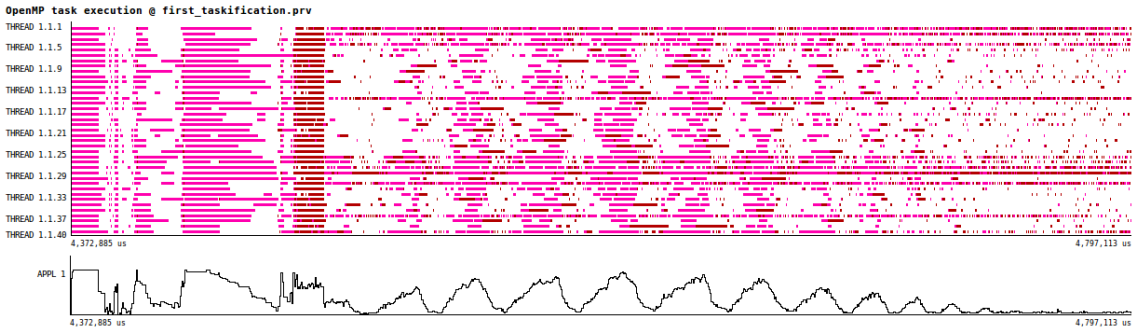

Figura 4.3: Primera Tasquificació: Cronograma d'execució de les tasques

En la Figura 4.4 podem observar la comparativa entre aquesta versió i l'*Original*. A l'eix x es representa el nombre d'OpenMP threads usats, mentre que a l'eix y es representa el speedup (guany en temps d'execució respecte dues execucions) obtingut amb l'execució sense OpenMP d'aquesta mateixa versió (*Primera Tasquificació*) com a referència. Amb un sol *thread*, aquesta versió té un speedup de 41,65$\times$ respecte l'*Original*, cosa que indica que l'*overhead* que introduien els *parallels* aniuats és molt significatiu. La diferència més gran s'obté amb 20 *threads* OpenMP, amb un speedup de 457,82$\times$. Pel que fa a la comparació

[1] La relació tamany-granularitat i codi de colors-traça de Paraver es repeteixen en les següents seccions, així que no es tornarà a indicar.

respecte de l'ideal, obté el seu millor valor per 20 *threads*, amb un 10,55×, fet que indica que hi ha marge de millora per les següents versions. D'altra banda, el speedup del codi original és bastant irrisori, amb valors al voltant de 0,045× a partir dels 4 *threads*.

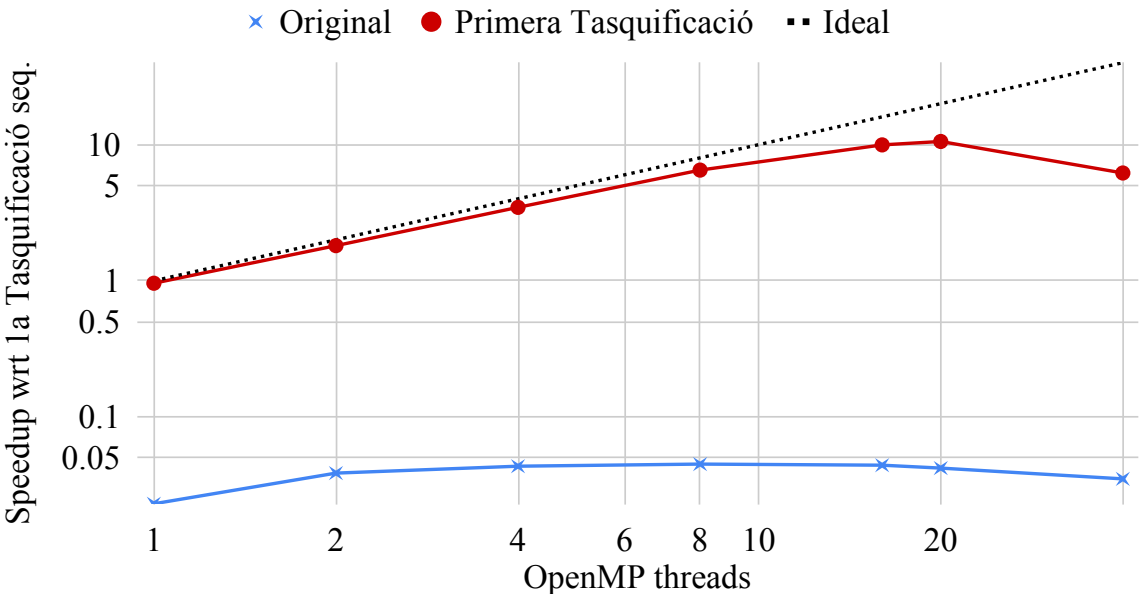

Figura 4.4: Primera Tasquificació vs Original

4.3 Tres Tasques

En la versió *Primera Tasquificació* s'ha observat un problema amb les tasques de granularitat molt fina, que introdueixen un *overhead* important. Com a solució a aquest problema, en aquesta versió es proposa una descomposició del programa en tasques una mica diferent, amb tres tipus de tasques: tasques de gra fi (línia 11), que s'encarreguen de realitzar tant els càlculs com la reducció de les parts de la matriu amb poca càrrega computacional; tasques de gra gruixut encarregades de la part de càlcul (línia 16); i la tasca encarregada de la seva corresponent reducció (línia 19). El programa generarà una sola tasca o dues en funció de la càrrega de treball que suposa aquell element de la Matriu Hamiltoniana i d'un *threshold* definit per l'usuari (línia 9); amb això s'evita la generació d'un nombre excessiu de tasques de granularitat petita. Per trobar el valor que ofereix el millor rendiment hem realitzat diverses execucions, intentant acotar els valors fins a arribar al desitjat. El codi corresponent es pot veure al Codi 4.2.

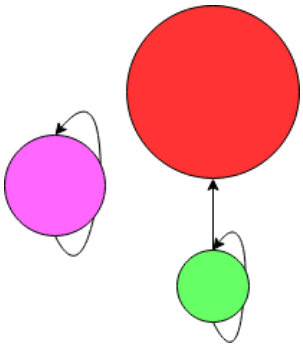

Figura 4.5: Dependències entre tasques: Tres Tasques

La Figura 4.5 mostra els diferents tipus de tasca i les dependències entre elles, de tal manera que el color *rosa* s'associa a les tasques de granularitat fina (línia 11), el *vermell* a les tasques de càlcul i el *verd* a les de reducció.

```
#pragma omp parallel
#pragma omp single
for(int ipatch=0; ipatch<npatches; ipatch++){
  for(int jpatch=0; jpatch<npatches; jpatch++){
    for(int k=0; k<CIJ.cij[ipatch][jpatch].size(); k++){
      Matrix A = CIJ.cij[ipatch][jpatch]->A[k];
      Matrix B = CIJ.cij[ipatch][jpatch]->B[k];
      if(A->nnz() && B->nnz()){
        if(vsize[ipatch] <= Threshold){
          //Create single task for small pieces of work
          #pragma omp task depend(inout: Y[i1:i2]) firstprivate(A, B)
          A->kron_mult('n','n', *A, *B, &X[j1], &Y[i1]);
        }else{
          //Create compute task and reduction task for larger pieces
          double* Y_tmp = new double[vsize[ipatch]]();
          #pragma omp task depend(inout: Y_tmp[0:vsize[ipatch]]) \
          firstprivate(A, B)
          A->kron_mult('n','n', *A, *B, &X[j1], &Y_tmp[0]);
          #pragma omp task depend(inout: Y[i1:i2]) \
          depend(in: Y_tmp[0:vsize[ipatch]])
          {
          int ilocal=0;
          for(int i=i1; i<i2; i++) Y[i] += Y_tmp[ilocal++];
          delete[] Y_tmp;
          }
        }
      }
    }
  }
}
```

Codi 4.2: Tres Tasques

En la Figura 4.6 podem observar una traça d'una execució d'aquesta versió. Amb aquesta estratègia l'aplicació és capaç d'explotar millor el paral·lelisme del qual disposa; a més a més, s'ha reduït l'*overhead* de la creació de tasques, ja que la mida mitjana de les tasques ha augmentat i el nombre total de tasques ha disminuït. La funció de la part inferior il·lustra com aquesta versió és capaç de fer un millor ús dels recursos disponibles, amb una reducció del nombre de pulsacions respecte a la Figura 4.3. No obstant això, també s'hi pot veure com encara hi ha zones en què bastants *threads* estan sense feina, fet que es podria millorar amb una millor planificació de les tasques.

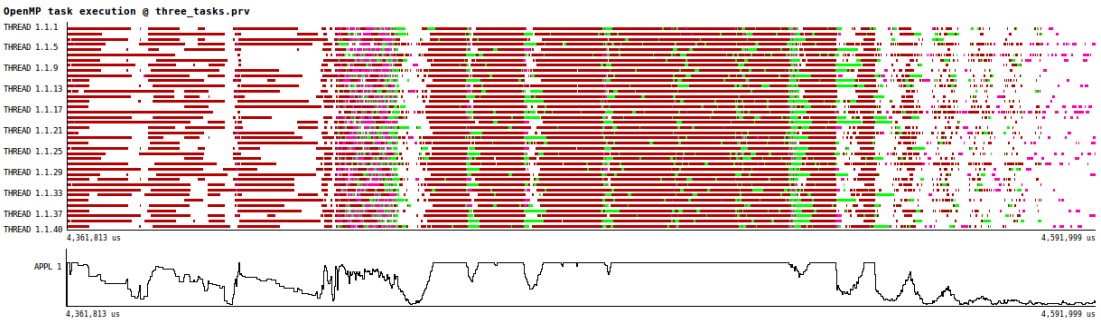

Figura 4.6: Tres Tasques: Cronograma d'execució de les tasques

En la Figura 4.7 es compara el speedup d'aquesta versió amb l'anterior (*Primera Tasquificació*), amb l'execució sense OpenMP de la *Primera Tasquificació* com a referència. Com es pot observar, el rendiment empitjora a l'executar el programa amb un sol *thread*, fruit de l'ús de l'estructura *if-else*. No obstant això, aquest fet ens permet millorar el rendiment a mesura que s'incrementa el nombre de *threads*, amb una millora de 1,19× amb 20 *threads* OpenMP.

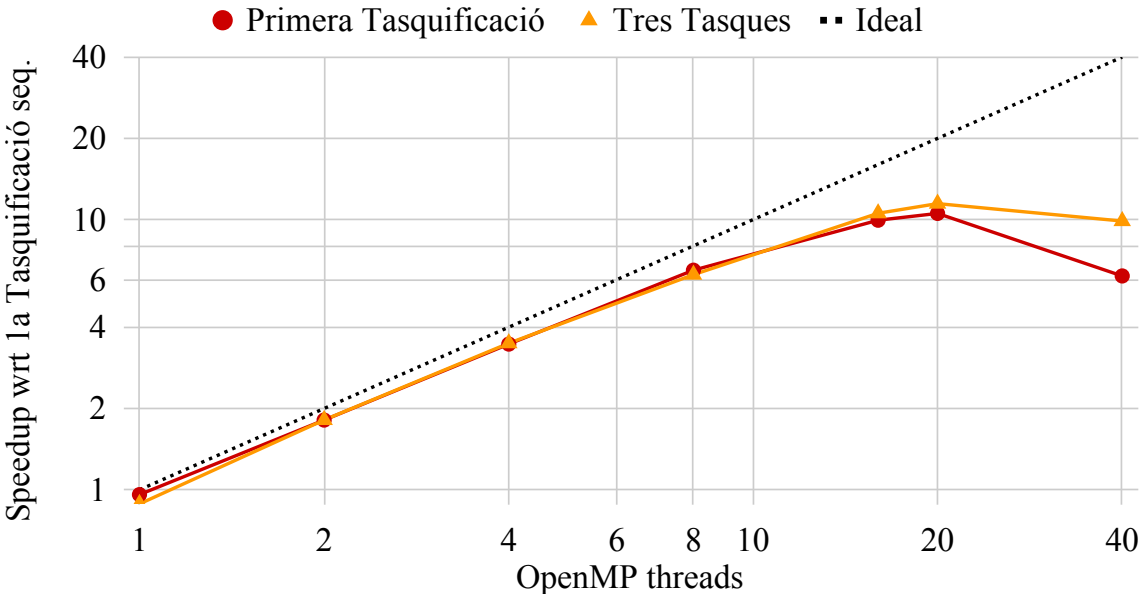

Figura 4.7: Tres Tasques vs Primera Tasquificació

4.4 Prioritats i Reutilització dels Buffers

En aquesta versió s'ataquen dos problemes de les dues versions ja vistes: a) Alt consum de memòria per culpa de la manera en què es fan els *mallocs* (consultar Secció 4.7) i b) Planificar les tasques adequadament per millorar el rendiment. El nou codi corresponent a aquesta versió es mostra al Codi 4.3. Per disminuir el consum de memòria, s'ha creat un vector (línia 1) que serveix per passar els *arrays* reservats amb memòria dinàmica (línia 20) d'una tasca a l'altra. Per mantenir la correctesa, s'estableix una antidependència en l'ús d'una posició del *buffer*, de manera que una tasca no pot escriure fins que l'anterior ha acabat, limitant el nombre de tasques de granularitat gruixuda que poden estar creades i llestes per ser executades a la vegada i, conseqüentment, el nombre màxim de vectors *Y_tmp*.

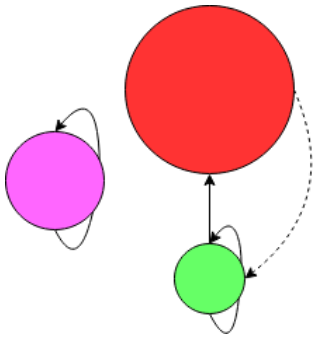

Figura 4.8: Dependències entre tasques: Prioritats

La Figura 4.8 mostra els diferents tipus de tasca i les seves dependències. En aquest cas, la distribució és exactament igual que en la versió anterior (veure Figura 4.5) i s'ha afegit l'antidependència amb la línia discontínua.

Pel que fa a la planificació de les tasques, se'ls hi ha afegit prioritats amb la intenció de millorar-la, intentant que les tasques amb més granularitat s'executin com més aviat millor i que s'alliberin els seus buffers. D'aquesta manera, a les tasques de computació (línia 17) se'ls hi assigna una prioritat variable, entre 0 i 1, en funció del seu volum de feina, relacionant directament feina a realitzar amb prioritat, cosa que fa que tinguin la mateixa prioritat que les tasques de gra fi i la seva execució es barregi. D'altra banda, les tasques de reducció (línia 24) tenen la màxima prioritat, amb la intenció de permetre a la següent tasca de càlcul executar-se com més aviat millor.

```
1  double* buffers[NBUFF]; //Set of buffers to limit memory usage
2  #pragma omp parallel
3  #pragma omp single
4  for(int ipatch=0; ipatch<npatches; ipatch++){
5    for(int jpatch=0; jpatch<npatches; jpatch++){
6      for(int k=0; k<CIJ.cij[ipatch][jpatch].size(); k++){
7        Matrix A = CIJ.cij[ipatch][jpatch]->A[k];
8        Matrix B = CIJ.cij[ipatch][jpatch]->B[k];
9        if(A->nnz() && B->nnz()){
10         if(vsize[ipatch] <= Threshold){
11           #pragma omp task depend(inout: Y[i1:i2]) \
12           firstprivate(A, B) priority(0)
13           kron_mult('n','n', A, B, &X[j1], &Y[i1]); //New call
14         }else{
15           mybuff = next = (next+1)%NBUFF;
16           int prio = vsize[ipatch] > PrioThreeshold; //Dynamic priority
17           #pragma omp task depend(inout:buffers[mybuff]) \
18             firstprivate(mybuff,ipatch,A,B) priority(prio)
19           {
20           double* Y_tmp = new double[vsize[ipatch]]();
21           buffers[mybuff] = Y_tmp;
22           kron_mult('n','n', A, B, &X[j1], Y_tmp);
23           }
24           #pragma omp task depend(inout:Y[i1:i2],buffers[mybuff]) \
25             firstprivate(mybuff) priority(10)
26           {
27           double* Y_tmp=buffers[mybuff];
28           int ilocal=0;
29           for(int i=i1; i<i2; i++) Y[i] += Y_tmp[ilocal++];
30           delete[] Y_tmp;
31           }
32         }
33       }
34     }
35   }
36 }
```

Codi 4.3: Prioritats i Reutilització dels Buffers

La Figura 4.9 mostra a la part superior una traça d'execució d'aquesta versió. Es pot observar com el *thread* 1, responsable d'instanciar les tasques, comença a executar tasques cap a la meitat de l'execució (s'aprecia millor en el zoom de la dreta), fet que significa que s'ha reduït substancialment l'*overhead* en la creació de tasques. A la part central de la figura es mostra l'ordre d'execució de les tasques, on el color verd representa a les tasques més velles (les que s'han instanciat primer) i el blau les més joves. S'hi observa, especialment en el zoom de la banda esquerra, com les tasques instanciades més tard (blau fosc) es

barregen amb tasques més velles (verd clar), fet que compacta molt més el cronograma, permet aprofitar molt més el paral·lelisme i eliminar les pulsacions que es veien abans (part inferior).

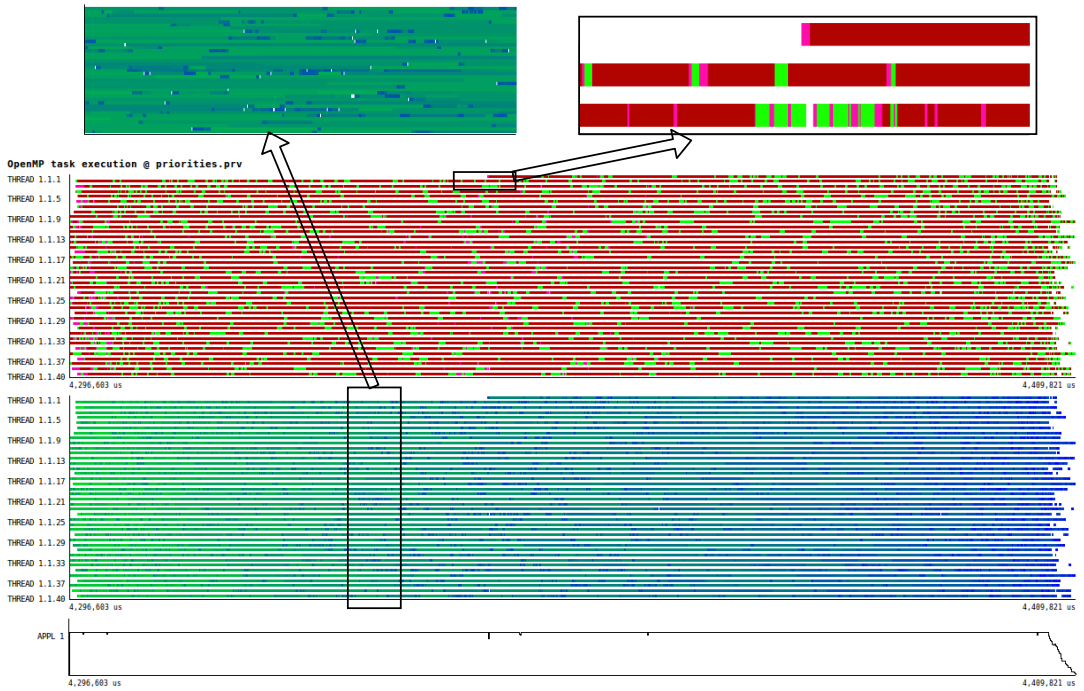

Figura 4.9: Prioritats i Reutilització dels Buffers:
Cronograma d'execució i ordre de les tasques

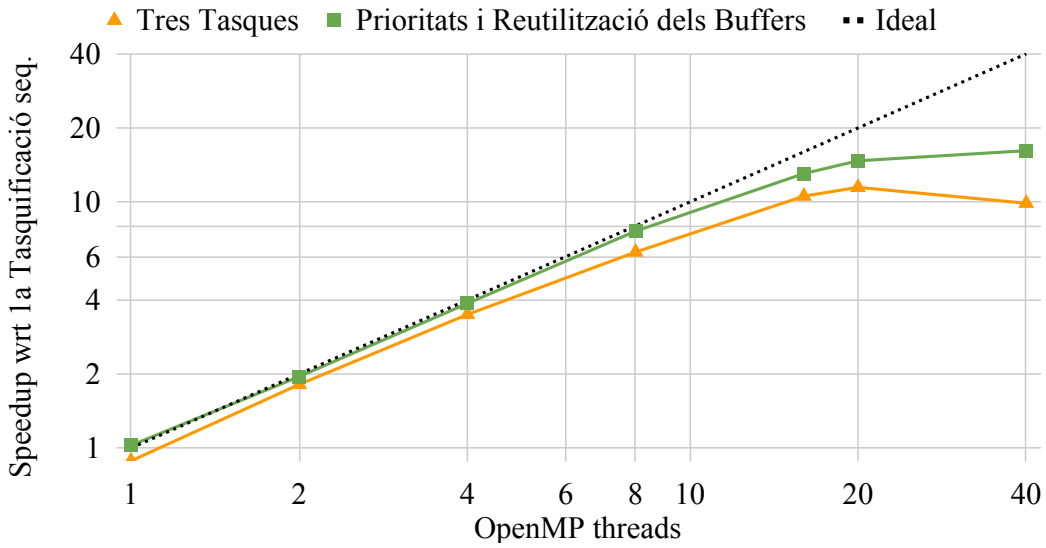

Figura 4.10: Prioritats i Reutilització dels Buffers vs Tres Tasques

La Figura 4.10 representa el speedup de la versió *Prioritats i Reutilització dels Buffers* comparat amb la versió anterior (*Tres Tasques*), amb l'execució sense OpenMP de la versió *Primera Tasquificació* com a referència. En aquesta ocasió, es pot apreciar com aquesta versió es comporta substancialment millor que les anteriors, sobretot a mesura que creix el

nombre de *threads*. Amb 16 *threads* OpenMP, el guany és de 1,28× respecte de la versió *Tres Tasques*. Quan escalem fins a 20 i 40 *threads* el rendiment també és superior al de la versió anterior, tot i que encara hi ha diferència amb l'ideal, sobretot per 40 *threads*. Tot i això, aquest és la primera versió en la qual el rendiment no empitjora per 40 *threads*.

4.5 Solapament d'Iteracions

Un dels problemes que presenta la versió *Prioritats i Reutilització dels Buffers* és el desbalanceig al final de l'execució, causat per una falta de paral·lelisme en aquell punt. Com ja s'ha vist a la Figura 3.4, l'aplicació original executa diverses vegades seguides aquest kernel, així que superposar l'execució de diverses iteracions és una manera de reduir aquest desbalanceig. Per aconseguir això, s'han mogut les directives *parallel* i *single* un nivell per sobre, com es pot veure al Codi 4.4. En aquest cas, no s'ha modificat l'estructura de les tasques, així que la seva representació és la mateixa que en la Figura 4.8.

```
#pragma omp parallel
#pragma omp single
for(int its=0; its<NITS; its++){
  for(int ipatch=0; ipatch<npatches; ipatch++){
    for(int jpatch=0; jpatch<npatches; jpatch++){
      for(int k=0; k<CIJ.cij[ipatch][jpatch].size(); k++){
        Matrix A = CIJ.cij[ipatch][jpatch]->A[k];
        Matrix B = CIJ.cij[ipatch][jpatch]->B[k];
        if(A->nnz() && B->nnz()){
          if(vsize[ipatch] <= Threshold){
            #pragma omp task depend(inout: Y[i1:i2]) \
            firstprivate(A, B) priority(0)
            kron_mult('n','n', A, B, &X[j1], &Y[i1]);
          }else{
            mybuff = next = (next+1)%NBUFF;
            int prio = vsize[ipatch] > PrioThreeshold;
            #pragma omp task depend(inout:buffers[mybuff]) \
            firstprivate(mybuff,ipatch,A,B) priority(prio)
            {
            double* Y_tmp = new double[vsize[ipatch]]();
            buffers[mybuff] = Y_tmp;
            kron_mult('n','n', A, B, &X[j1], Y_tmp);
            }
            #pragma omp task depend(inout:Y[i1:i2],buffers[mybuff]) \
            firstprivate(mybuff) priority(10)
            {
            double* Y_tmp=buffers[mybuff];
            int ilocal=0;
            for(int i=i1; i<i2; i++) Y[i] += Y_tmp[ilocal++];
            delete[] Y_tmp;
            }
          }
        }
      }
    }
} }
```

Codi 4.4: Solapament d'Iteracions

En la Figura 4.11 podem veure una traça d'aquesta versió, que inclou 5 iteracions. S'ha escollit aquest número perquè permet mostrar com s'entrellacen les execucions de diferents iteracions i alhora no és molt gran, cosa que no incrementa la complexitat del gràfic. Pel que fa als colors de la imatge, es mantenen els mateixos que en les dues versions anteriors: rosa per les tasques de granularitat fina encarregades dels càlculs i la reducció, vermell per les de granularitat gran a càrrec de la computació, i verd per les de reducció. Tot i que podem diferenciar les diferents iteracions visualment, es pot comprovar que les tasques que pertanyen a diferents iteracions s'executen concurrentment, cosa que elimina els desbalancejos periòdics al final de cada iteració i ajuda a incrementar el paral·lelisme.

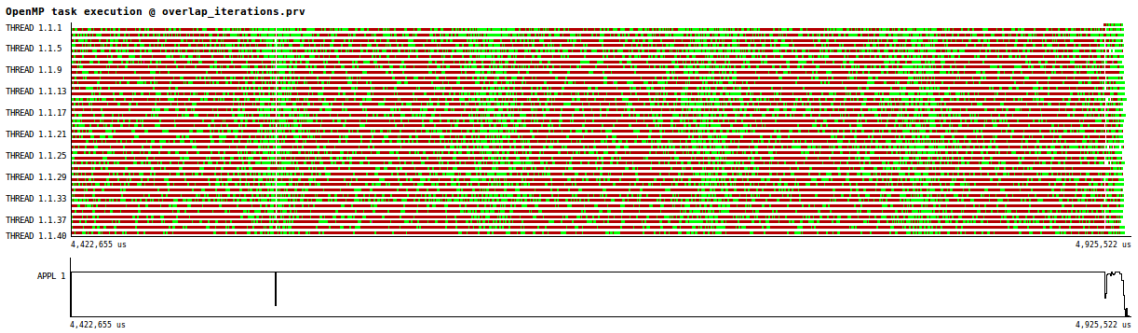

Figura 4.11: Solapament d'Iteracions: Cronograma d'execució de les tasques

La Figura 4.12 mostra el speedup de la versió *Solapament d'Iteracions*. Com es pot veure, aquesta versió es comporta lleugerament millor que la versió anterior (*Prioritats i Reutilització dels Buffers*), sense una diferència clara, excepte en el cas de 40 *threads* OpenMP, on s'obté un guany de 1,24×. Això és degut al fet que el grau de desbalanceig augmenta a mesura que s'usen més *threads*, així que a mesura que anem escalant traurem un millor profit de solapar iteracions.

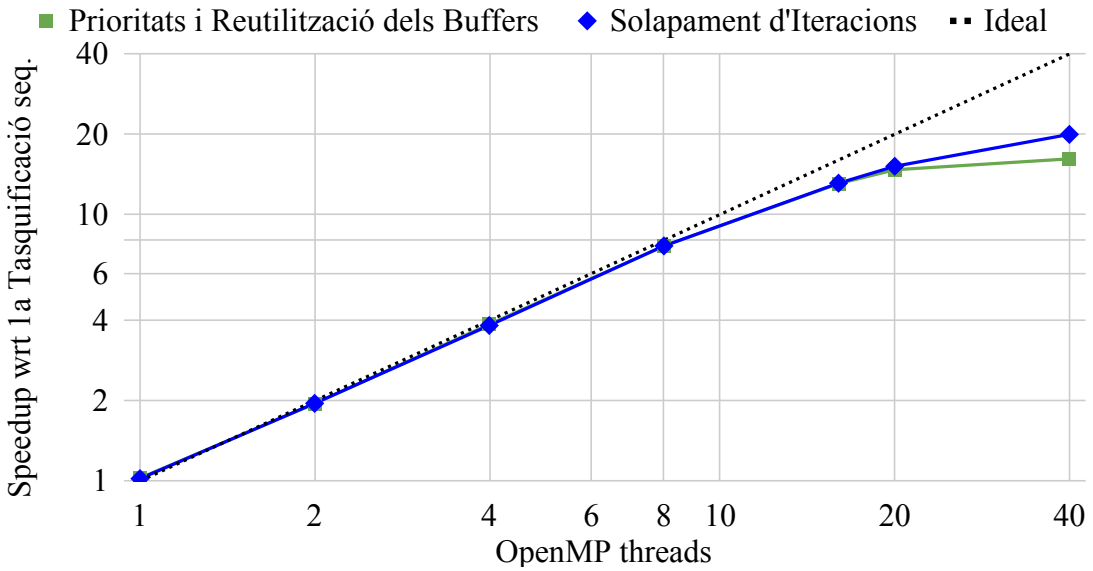

Figura 4.12: Solapament d'Iteracions vs Prioritats i Reutilització dels Buffers

4.6 Tasques Aniuades

Analitzant amb detall la versió *Solapament d'Iteracions* es pot veure que la granularitat de les tasques roses, amb una duració mitjana d'uns pocs microsegons, està limitant l'escalabilitat de l'aplicació. Per solucionar aquest problema, es dissenya una nova estructura de tasques pel programa, amb dos nivells de tasques. Aquesta estratègia és possible perquè cada iteració del *for* d'*ipatch* és independent de les altres i el volum de feina a realitzar ve determinat per la mateixa variable *ipatch*. Podem veure els canvis aplicats en el Codi 4.5.

```
1  char* sentinel = new char[npatches](); //Dependence token
2  #pragma omp parallel
3  #pragma omp single
4  for(int its=0; its<NITS; its++){
5    for(int ipatch=0; ipatch<npatches; ipatch++){
6      int fine_grain = vsize[ipatch] <= Threshold;
7      #pragma omp task depend(inout: sentinel[ipatch]) priority(10)
8      for(int jpatch=0; jpatch<npatches; jpatch++){
9        for(int k=0; k<CIJ.cij[ipatch][jpatch].size(); k++){
10         Matrix A = CIJ.cij[ipatch][jpatch]->A[k];
11         Matrix B = CIJ.cij[ipatch][jpatch]->B[k];
12         if(A->nnz() && B->nnz()){
13         if(fine_grain){ //Each task will always take the same path
14           kron_mult('n','n', A, B, &X[j1], &Y[i1]);
15         }else{
16           double** buffer = new double*;
17           #pragma omp task depend(out:buffer) \
18           firstprivate(A,B,buffer,ipatch)priority(0)
19           {
20           double* Y_tmp = new double[vsize[ipatch]]();
21           buffer[0] = Y_tmp;
22           kron_mult('n','n', A, B, &X[j1], Y_tmp);
23           }
24           #pragma omp task depend(inout:Y[i1:i2])depend(in:buffer) \
25           firstprivate(buffer)priority(5)
26           {
27           double* Y_tmp=buffer[0];
28           int ilocal=0;
29           for(int i=i1; i<i2; i++)
30             Y[i] += Y_tmp[ilocal++];
31           delete[] Y_tmp;
32           delete[] buffer;
33           }
34           }
35         }
36       }
37     }
38   #pragma omp taskwait
39   }
40 }
```

Codi 4.5: Tasques Aniuades

Per cada iteració del bucle *ipatch* es genera una tasca (línia 7), que pot dirigir-se per dos camins diferents, en funció del volum de feina que comporta aquell *ipatch* i un llindar definit per l'usuari (línia 6). En cas que la quantitat de feina no sigui gran, la mateixa tasca realitzarà els càlculs i les reduccions en sèrie (línia 14), sense generar cap tasca; en cas contrari, si es determina que el volum de feina és elevat, es generaran dues tasques per cada element, de manera similar a les versions anteriors, una per la computació (línia 17) i una altra per la seva reducció (línia 24). Per garantir que els *ipatch* de diferents iteracions s'executen en l'ordre adequat, és a dir, que no s'avancen entre ells, es fa servir un sentinella (línia 1) per establir aquesta dependència en la primera tasca, de tipus *inout*. Aquest element ens genera una dependència *falsa*, que ens assegura que el programa s'executa com nosaltres volem.

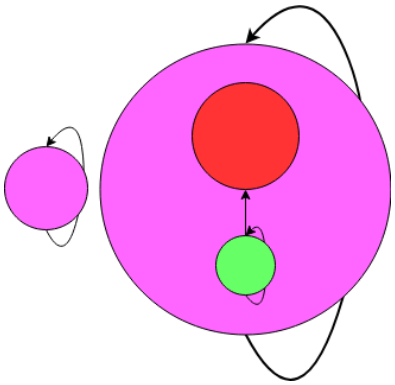

Figura 4.13: Dependències entre tasques: Tasques Aniuades

La Figura 4.13 representa l'estructura de tasques i dependències descrita. Hi podem veure dibuixats els dos camins descrits: a l'esquerra, una tasca de color *rosa* més petita, que es correspon amb el camí de l'*if* del Codi 4.5, i una tasca de color *rosa* gran que represente l'*else*. Dins d'aquesta última s'hi representen una tasca de color *vermell*, que fa referència a les tasques de càlcul, i una de color *verd*, corresponent a les de reducció.

La Figura 4.14 ens mostra una traça de Paraver d'aquesta versió, amb 5 iteracions solapades. Aquesta versió aconsegueix reduir el nombre total de tasques creades i paral·lelitza la seva instanciació, cosa que redueix l'*overhead* de la versió *Solapament d'Iteracions*. Com a contrapart, presenta un notable desbalanceig al final, a causa de la instanciació de tasques al final de l'execució. Aquest fet fa que en la recta final no hi hagi suficients tasques per omplir tots els *cores* disponibles, limitant el rendiment d'aquesta versió.

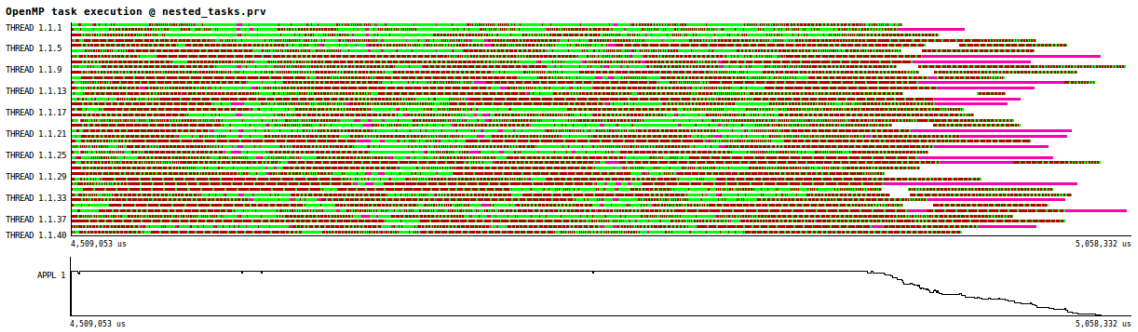

Figura 4.14: Tasques Aniuades: Cronograma d'execució de les tasques

A la Figura 4.15 hi podem observar la comparació del speedup entre aquesta versió i la de *Solapament d'Iteracions*. Quan es fan servir 8 i 16 *threads* OpenMP hi ha un lleuger guany, de 1,06× i 1,05×, respectivament; amb 40 *threads* el guany que s'obté és encara menor, al voltant de 1,03×, per culpa del gran desbalanceig al final de l'execució, com s'ha vist a la Figura 4.14. De totes maneres, aquesta versió és la que té un millor rendiment i, com es veurà a la Figura 4.16 de la següent secció, també és la que consumeix menys memòria.

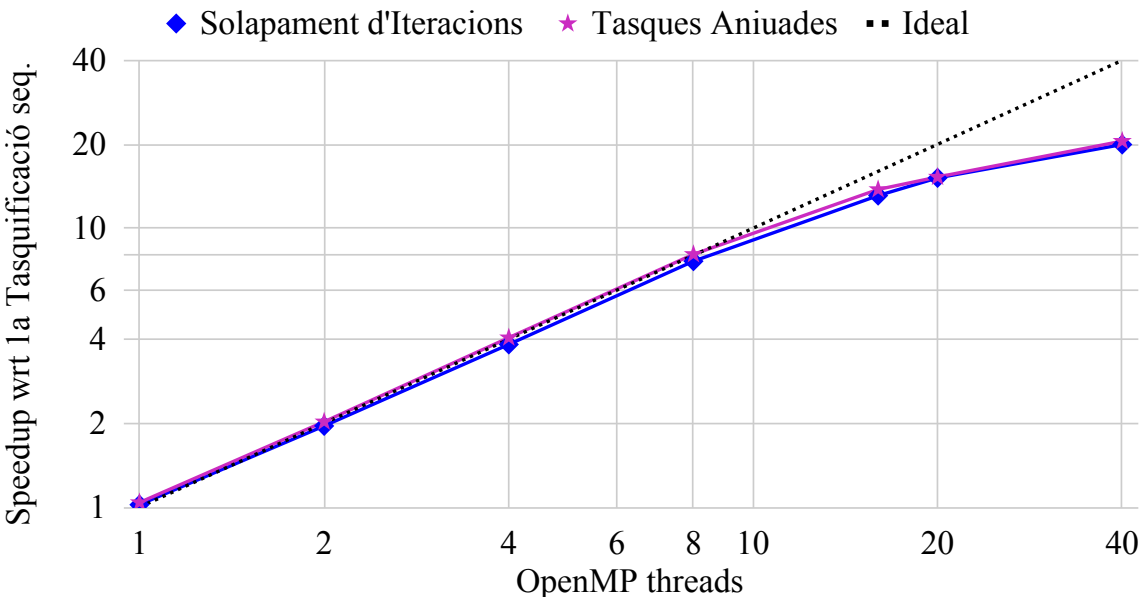

Figura 4.15: Tasques Aniuades vs Solapament d'Iteracions

4.7 Consum de Memòria

Com ja s'ha introduït amb anterioritat, en aquest projecte no buscàvem solament reduir el temps d'execució, sinó també el consum de memòria de l'aplicació. Tot i que en el cas de la *miniapp* el consum de memòria no és un element crític, sí que ho és pels científics que usen l'aplicació habitualment, i reduir el consum els permetrà utilitzar *inputs* més grans en les seves simulacions. A més a més, d'aquesta manera queda demostrat que aquestes tècniques milloren el temps d'execució sense perjudicar el consum de memòria. En la Figura 4.16 es pot veure l'evolució del consum de memòria al llarg de tot el projecte i per tots els nombres de *threads* OpenMP fets servir. El valor que es mostra és el consum pic que ha tingut el programa, sempre expressat en GB.

Consum de Memòria [GB]	**OpenMP Threads**						
Versió del codi	1	2	4	8	16	20	40
Original	1.39	1.40	1.40	1.42	1.46	1.47	1.55
Primera Tasquificació	1.39	1.43	1.44	1.47	1.62	1.64	1.61
Tres Tasques	1.39	1.43	1.45	1.52	1.64	1.61	1.61
Prioritats i Reutilització dels Buffers	1.39	1.40	1.41	1.42	1.44	1.46	1.50
Solapament d'Iteracions	1.39	1.41	1.41	1.47	1.47	1.48	1.49
Tasques Aniuades	1.39	1.39	1.40	1.41	1.43	1.41	1.46

Figura 4.16: Consum de memòria màxim en GB per cada versió i número de *threads*

Les dues primeres versions, tot i millorar el temps d'execució de l'*Original*, ho feien a costa de gastar més memòria, consumint uns 140~170 MB més amb 20 *threads* OpenMP. Com s'explica a la Secció 4.4, aquest increment en el consum ve donat per una mala estratègia en reservar memòria dinàmica i es pot evitar canviant unes poques línies de codi. Amb

els canvis introduïts a la versió *Prioritats i Reutilització dels Buffers* es soluciona aquest problema, reduint el consum en 150 MB respecte a la versió *Tres Tasques*. En pujar un nivell les directives *parallel* i *single* en la versió *Solapament d'iteracions* s'incrementa el consum lleugerament, en 20 MB, respecte de la versió anterior. En la darrera versió, *Tasques Aniuades*, s'aconsegueix reduir el consum en 60 MB respecte l'*Original* i en 70 MB en comparació amb la versió *Solapament d'Iteracions*, aconseguint el consum més baix en tot el projecte. Aquest fet és possible gràcies a la nova descomposició en tasques que s'implementa, que redueix el nombre de tasques creades i llestes per ser executades i acaba fent disminuir el consum pic del programa.

4.8 Comparativa Final

Finalment, la Figura 4.17 inclou una comparativa entre totes les versions creades, amb l'objectiu de poder apreciar les diferències entre totes les versions a la vegada, en contrapartida a les comparatives fetes fins ara. S'ha deixat la versió *Original* fora a causa de la seva gran diferència en temps d'execució amb la resta, cosa que ens permet apreciar millor els valors del gràfic. Si es vol comparar amb l'*Original*, la Figura 4.4 ja ofereix una comparativa prou bona. En aquest cas, l'estructura dels eixos és la mateixa que en les seccions anteriors, amb l'eix x representant el nombre de *threads* OpenMP, i l'eix y representant el speedup respecte de la versió *Primera Tasquificació* executada sense OpenMP.

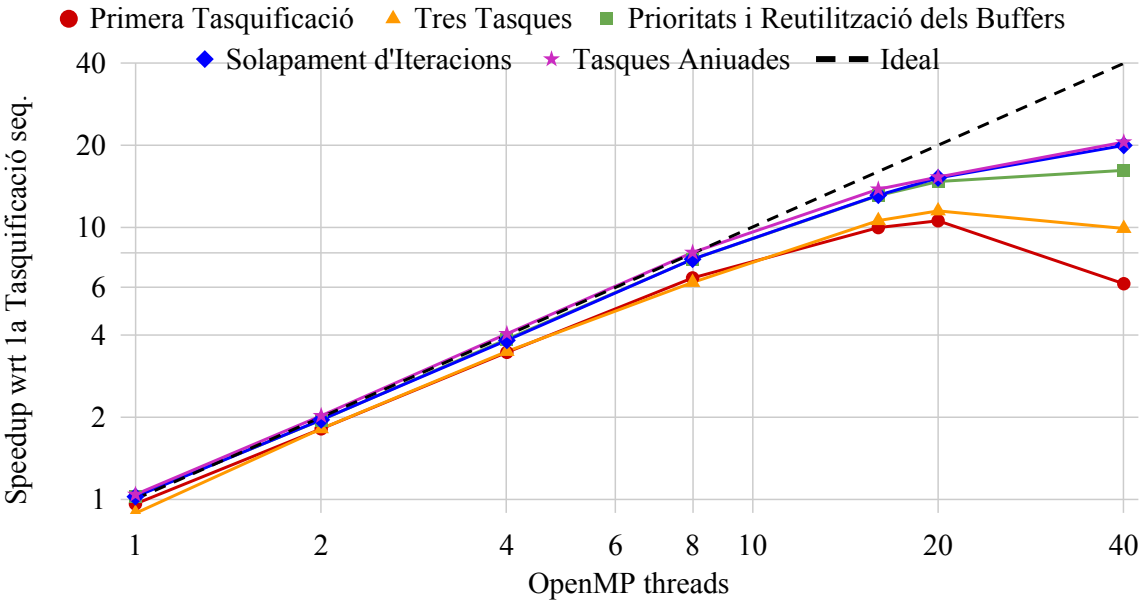

Figura 4.17: Comparativa final entre totes les versions

Inicialment, la versió *Primera Tasquificació* no tardava a allunyar-se del speedup ideal, amb un valor de 6,55× per 8 *threads* OpenMP. Amb les modificacions aplicades a cada versió, s'ha aconseguit que la versió *Tasques Aniuades* es mantingui sobre el rendiment ideal, amb un speedup de 8,06×, 1,24× millor que la *Primera Tasquificació*. Per 16, 20 i 40 *threads* OpenMP totes les versions s'allunyen progressivament del rendiment ideal, tot i que hi ha una millora clara de les dues últimes versions en comparació amb les primeres, amb una diferència de 3,32× entre les versions *Primera Tasquificació* i *Tasques Aniuades* en usar 40 *threads*. La diferència entre les versions *Original* i *Tasques Aniudaes* arriba fins a 585× per 40 *threads*, però és una comparació bastant injusta, a causa del mal rendiment que

ofereixen els *parallel for* aniuats.

Les pèrdues de rendiment les podem atribuir a diversos motius. Primer de tot, es pot veure al llarg de totes les versions que en el canvi entre 20 i 40 *threads* el rendiment empitjora, es manté igual o es guanya molt poc. Tal com estan dissenyats els nodes de CTE-Power, el canvi de 20 a 40 correspon amb el pas d'usar un sol socket a fer-ne servir dos i, com es tracta d'una arquitectura NUMA (**N**on-**U**niform **M**emory **A**ccess), el temps d'accés a la memòria de l'altre socket és major, cosa que perjudica el rendiment perquè tota la memòria usada correspon a la del primer socket. A més a més, altres factors ja vists abans, com l'*overhead* introduït per la creació de tasques i l'estructura *if-else* i el desbalanceig final (especialment en la versió *Tasques Aniuades*) també contribueixen a disminuir l'escalabilitat de l'aplicació.

Finalment, cal comentar que s'ha decidit no seguir fent millores en aquest punt per diversos motius. Primer de tot, el temps necessari per implementar una nova versió ha anat augmentant progressivament al llarg del projecte (amb l'excepció de la versió *Solapament d'Iteracions*). Les dates d'entrega del TFG i d'un article científic (vegeu Secció 5.9) també han afavorit a això, sumat al fet que s'ha considerat que el volum de la feina realitzada era suficientment elevat.

Capítol 5

Planificació, Gestió Econòmica i Desviacions Respecte la Planificació Inicial

El temps aproximat per la realització d'aquest projecte és d'uns cinc mesos, iniciant-se el febrer de 2019 i finalitzant la primera setmana de juliol de 2019 amb la presentació final. A continuació es detallen les tasques planificades i els recursos necessaris per assolir els objectius del projecte, així com les solucions proposades davant les possibles desviacions que podrien aparèixer durant el projecte. Seguidament, es tracten els aspectes relacionats amb el pressupost de la planificació inicial. Finalment, es tanca el capítol explicant com han variat la planificació i el pressupost inicial, un cop el projecte ja es troba en el seu punt final.

5.1 Descripció de les tasques

5.1.1 Gestió del Projecte (75 hores)

Aquesta tasca es correspon amb l'assignatura de GEP, emmarcada dins del TFG de la FIB. Consta d'un total de 75 hores, repartides entre les següents subtasques:

- **Recerca i Estudi de Referències**: 14,75 hores.
- **Abast del Projecte i Contextualització**: 24,5 hores.
- **Planificació Temporal**: 8,25 hores.
- **Gestió Econòmica i Sostenibilitat**: 9,25 hores.
- **Presentació Oral i Document final**: 18,25 hores.

Per a realitzar aquesta tasca serà necessari l'ús d'un ordinador portàtil, un cercador d'articles com Google Scholar, Google Docs per a realitzar el redactat i la presentació, l'eina Gantter per elaborar la planificació i El Racó i Atenea.

5.1.2 Estudi del codi i Anàlisi inicial (30 hores)

Abans de començar a millorar el codi és molt important dedicar un temps a estudiar-lo i entendre'l. En aquesta tasca cal familiaritzar-se amb les estructures de dades i el *workflow*

del programa, aprendre a compilar-lo i executar-lo, i crear scripts per facilitar aquestes tasques i l'obtenció de mètriques i traces. També, és el punt per prendre decisions que marcaran el rumb del projecte, com ara el model de programació i l'arquitectura a utilitzar. Per acabar, aprofitant les traces obtingudes es pot realitzar la primera anàlisi del programa. En total, hem estimat la durada d'aquesta tasca en 30 hores.

En aquesta tasca serà necessari l'ús d'un ordinador portàtil, l'accés al clúster CTE-Power i les seves llibreries, Skype, i les eines d'anàlisi Extrae i Paraver.

5.1.3 Procés iteratiu de millores (300 hores)

Com s'ha explicat a la Secció 2.3, aquest projecte se centra en aplicar un procés iteratiu per a millorar una aplicació. Així doncs, en aquest punt trobem una tasca gran, dividida en petites subtasques, que s'anirà repetint fins a assolir l'objectiu desitjat. A causa del mètode de treball, és difícil establir una durada determinada per aquesta tasca, però s'ha estimat la durada en 300 hores dividides entre totes les subtasques. Per a ser flexibles amb la planificació, com es pot observar en el Diagrama de Gantt, tenim temps al final del projecte per estendre aquesta tasca i seguir complint amb la data límit d'entrega.

Per dur a terme aquesta tasca serà necessari l'ús d'un ordinador personal, l'accés al clúster CTE-Power del BSC, les eines d'anàlisi Extrae, Paraver i Valgrind, Google Drive/Docs per a redactar i elaborar taules, Skype per a qualsevol videoconferència, i Vim per editar el codi. A continuació s'exposa individualment cada una de les subtasques; cal tenir en compte que els temps designats a cada una d'elles és orientatiu i una pot acabar tenint-ne més i una altra menys.

5.1.3.1 Introducció de millores i verificació del resultat (120 hores)

Es tracta de la primera subtasca de la iteració de millores. En aquesta part del projecte s'aprofitarà l'anàlisi realitzada a l'iteració anterior (o bé l'inicial) per introduir les millores en el codi. Un cop fet això, és molt important verificar que el resultat segueix sent correcte, comparant l'*output* de la versió actual amb l'Original amb l'eina de Linux *diff*. Quan s'han afegit tots els canvis al codi i comprovat el resultat es pot avançar a la següent tasca. En total, hem estimat la durada d'aquest apartat en 120 hores, al llarg de totes les iteracions.

5.1.3.2 Obtenció de mètriques (60 hores)

Aquesta subtasca de la iteració estarà dedicada a l'obtenció del consum de memòria i temps d'execució de la versió actual de l'aplicació. A causa del nombre de cores de la màquina, hem decidit prendre les mesures per 1, 2, 4, 8, 16, 20 (coincideix amb el nombre de cores per socket) i 40 (nombre total de cores físics en un node del clúster).

En alguna iteració pot aparèixer algun paràmetre addicional en l'aplicació, així que també pot ser necessari prendre mesures per diferents valors d'aquest atribut. Un cop es tenen totes les mètriques és quan es pot avaluar realment l'impacte de la millora en l'aplicació. El temps que hem estimat per aquesta tasca és de 60 hores, al llarg de totes les iteracions.

5.1.3.3 Anàlisi de rendiment (120 hores)

Es tracta de l'última subtasca de la iteració de millores. Aquí obtindrem traces de l'execució del programa amb Extrae per després analitzar-les amb Paraver i, si fos necessari,

faríem ús d'algun profiler com ara Valgrind. Un cop realitzat l'anàlisi es podrà decidir els següents passos d'optimització o bé es decidirà que aquest procés ja ha arribat al punt final i es procedirà a la següent tasca. Per aquesta subtasca hem estimat una durada de 120 hores, al llarg de totes les iteracions.

5.1.4 Avaluació final del rendiment (20 hores)

Un cop acabada la iteració de millores caldrà prosseguir amb aquesta tasca, en la qual es recolliran les dades finals del projecte i es farà una avaluació de l'evolució de l'aplicació al llarg del projecte, elaborar gràfiques i taules, etc. Per a dur a terme aquesta tasca seran necessàries unes 20 hores de treball. Com a recursos, farem ús dels mateixos que a la tasca anterior.

5.1.5 Documentació i Tancament del Projecte (65 hores)

Per acabar el projecte serà necessari elaborar la documentació necessària i tancar el projecte. Aquest últim pas inclou qualsevol comunicació i reunió amb l'equip d'ORNL, així com la redacció de la memòria del TFG i la preparació de la presentació. El temps estimat per aquesta tasca és de 65 hores.

Els recursos necessaris en aquest punt del projecte seran un ordinador portàtil, Skype, Vim per editar qualsevol codi puntualment, el programari de Google Drive i TeXstudio per donar un acabat més professional a la memòria.

5.2 Previsió Temporal

En la següent taula es mostra el temps previst per cada tasca, així com el temps total pel projecte. En el Diagrama de Gantt (Figura 5.1) també es pot apreciar en certa manera, tant això com les dependències explicitades en la Secció 5.3. Com es podrà veure allà, el projecte està previst que acabi el 18 de juny, planificant cada jornada amb cert marge i deixant uns dies al final de coixí per qualsevol imprevist. Finalment, cal dir que cada jornada de treball serà de sis hores, generalment de 8.00 a 14.00.

Tasca	**Dedicació (hores)**
1. Gestió del Projecte	75
2. Estudi del codi i Anàlisi inicial	30
3. Procés iteratiu de millores	300
–>3.1. Introducció de millores i verificació del resultat	120
–>3.2. Obtenció de mètriques	60
–>3.3. Anàlisi de rendiment	120
4. Avaluació final del rendiment	20
5. Documentació i tancament del projecte	65
Total	**490**

Taula 5.1: Durada de les tasques

Tasca	Tasca predecessora
1. Gestió del Projecte	-
2. Estudi del codi i Anàlisi inicial	1. Gestió del Projecte
3. Procés iteratiu de millores	2. Estudi del codi i Anàlisi inicial
–>3.1. Introducció de millores i verificació del resultat	2. Estudi del codi i Anàlisi inicial
–>3.2. Obtenció de mètriques	3.1. Introducció de millores i verificació del resultat
–>3.3. Anàlisi de rendiment	3.2. Obtenció de mètriques
4. Avaluació final del rendiment	3. Procés iteratiu de millores
5. Documentació i tancament del projecte	4. Avaluació final del rendiment

Taula 5.2: Dependències entre tasques

5.3 Dependències entre tasques

5.4 Diagrama de Gantt

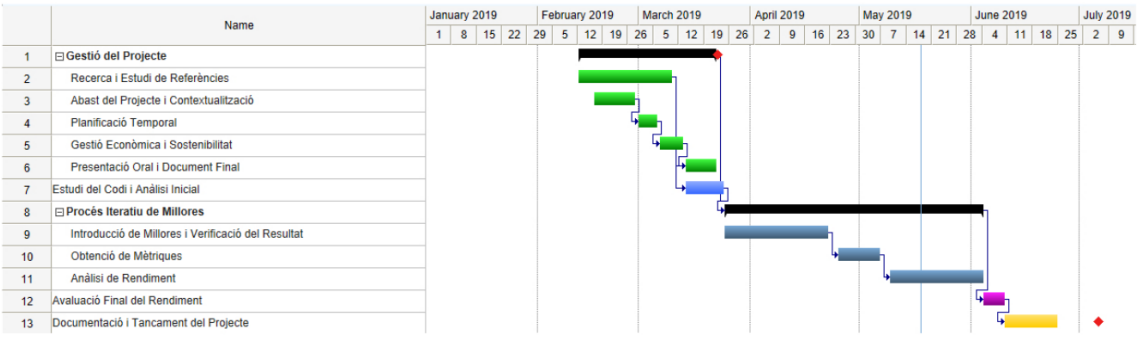

Figura 5.1: Diagrama de Gantt

5.5 Recursos

A continuació es resumeixen el conjunt de recursos, de qualsevol classe, necessaris per a dur a terme aquest projecte.

5.5.1 Recursos Hardware

- Ordinador portàtil. Es tracta d'un Dell Latitude 7480, amb 16 GB de RAM i un processador Intel Core i7-7600U 2,80 GHz.
- Monitor, teclat i ratolí. Per millorar les condicions de feina i guanyar comoditat es faran servir aquests recursos addicionals.
- Clúster CTE-Power. Es tracta d'un supercomputador del BSC basat en l'arquitectura IBM Power9. Té 52 nodes de còmput amb dos processadors IBM Power9 [27]

8335-GTH 2,4 GHz, 512 GB de RAM, quatre GPUs NVIDIA Volta V100 i una xarxa d'interconnexió Infiniband.

5.5.2 Recursos Software

- Paraver: Visualitzador de traces d'execució.
- Extrae: Generació de traces de Paraver.
- Valgrind: Profiler.
- Vim: Editor de Text.
- Skype: Programa per a realitzar videoconferències.
- Gantter: Eina en línia per crear diagrames de Gantt.
- Google Drive: Emmagatzematge de fitxers en línia i editor de text, fulls de càlcul i presentacions.
- TeXstudio: IDE de LaTeX.
- Mozilla Thunderbird: Gestor de correus electrònics, comunicació amb la resta d'integrants de l'equip.
- Racó i Atenea: Documentació del projecte, entregues i coordinació amb professors.

5.5.3 Recursos Humans

En aquest projecte hi ha un seguit de persones implicades, com s'indica a la Secció 2.4. El desenvolupador serà el responsable principal de la feina, mentre que la resta tenen un rol més enfocat al suport.

5.6 Possibles alteracions i alternatives

És probable que durant el transcurs del projecte es produeixin alteracions a la planificació inicial, així que cal pensar prèviament com solucionar aquestes situacions. Com s'indica a la Secció 2.3 Metodologia, durant tot el projecte treballarem basant-nos en la metodologia SCRUM, que ens ofereix una gran llibertat i ens permet adaptar-nos fàcilment a aquestes situacions, adaptant la planificació sobre la marxa en funció dels requeriments d'un moment determinat. A més a més, a l'hora d'elaborar aquesta planificació ja s'ha tingut en compte l'aparició de desviacions, així que hi ha marge suficient per a adaptar-la enmig del projecte.

Podem trobar dues situacions completament oposades:

- Una tasca acaba abans de temps. Aquesta situació no suposa cap inconvenient, ja que no implica cap desajust de la planificació. En aquest cas simplement podem procedir amb la pròxima tasca abans.
- Una tasca s'allarga més de l'esperat. En aquestes situacions cal allargar la durada de la tasca en qüestió i postposar l'inici de la següent. Com hem dit anteriorment, la planificació està pensada perquè es pugui donar aquesta situació i seguir complint

la data d'entrega. En un cas extrem, si es veu que no hi ha temps suficient, es pot tallar el procés iteratiu de millores sense arribar a l'objectiu final i procedir amb la següent tasca.

D'aquesta manera, podem assegurar el compliment de totes les tasques planificades en la seva totalitat o, en un cas extrem, en la seva majoria. A més a més, qualsevol variació en la planificació inicial no afectaria els recursos necessaris en cap cas, tot i que sí que podria tenir un impacte en el pressupost.

5.7 Pressupost

A l'hora d'elaborar el pressupost del projecte s'ha fet una diferenciació entre els costos indirectes i els directes, desglossant aquests últims segons les tasques de la planificació.

Per elaborar les despeses directes, hem tingut en compte tot el material descrit en cada tasca i el salari del desenvolupador principal, extret de la seva nòmina. A causa que tot el software utilitzat és gratuït, només s'han tingut en compte salaris i despeses de *hardware* en aquest punt. Per establir el cost €/h del clúster CTE-Power s'ha tingut en compte que només s'usaran 2 nodes simultàniament durant el projecte, un d'ells com a node de login i l'altre per a càlcul. El seu preu total ha estat estimat a partir del cost del supercomputador Summit, en el qual s'inspira el CTE-Power.

En les despeses indirectes hem agrupat els salaris de la resta de personal relacionat en menor manera amb el projecte (directora, codirector, ponent...), ja que no es pot imputar directament a una tasca. S'ha estimat la seva contribució en 100 hores, entre tots, al llarg del projecte. El preu per hora s'ha estimat partint del sou del Desenvolupador, tenint en compte que ocupen posicions superiors i tenen una jornada laboral de dues hores més. Pel que fa a les "despeses d'oficina", aquí s'engloba l'ús del despatx ubicat al K2M, la línia telefònica, Internet, electricitat, calefacció, aigua i neteja. Cal destacar que el preu de tots aquests elements junts és superior al mostrat, però com el despatx és compartit no es pot atribuir tot a un sol projecte.

Com es va indicar a la planificació, la tasca "Procés iteratiu de millores" és una gran candidata a sofrir desajustos, i per tant, pot durar més del que s'ha previst. Per això s'ha afegit una despesa de 618,4 € al 20% com a imprevist en aquesta tasca, que corresponen al salari del desenvolupador i ús de l'ordinador durant 60 hores extres i l'ús del clúster durant 10 hores addicionals. La resta veurien cobert qualsevol desajust amb el pressupost de contingència, que s'ha establert en un 15% del total, ja que fàcilment podem requerir més hores de computació o bé més hores per acabar una tasca.

	Unitats (Producte o hores)	Preu unitat (€)	Vida útil (anys)	Amortització (€/h)	Preu (€)
Costos directes					**5.165,28**
Gestió del Projecte	**75 hores**				**768,00**
Ordinador (Portàtil, Pantalla, Teclat i Ratolí)	1,00	1.900,00	4,00	0,24	18,00
Google Software	1,00	0,00	-	0,00	0,00

	Unitats (Producte o hores)	Preu unitat (€)	Vida útil (anys)	Amortització (€/h)	Preu (€)
Gantter	1,00	0,00	-	0,00	0,00
Racó + Atenea	1,00	0,00	-	0,00	0,00
Desenvolupador	1,00	10,00	-	-	750,00
Estudi del codi i Anàlisi inicial	**30 hores**				**3.215,68**
Ordinador	1,00	1.900,00	4,00	0,24	7,20
Extrae	1,00	0,00	-	0,00	0,00
Paraver	1,00	0,00	-	0,00	0,00
Skype	1,00	0,00	-	0,00	0,00
Clúster CTE-Power	1,00	2.000.000	4,00	0,40	2,00
Desenvolupador	1,00	10,00	-	-	300,00
Procés iteratiu de millores	**300 hores**				**3.215,68**
Ordinador	1,00	1.900	4,00	0,24	72,00
Extrae	1,00	0,00	-	0,00	0,00
Paraver	1,00	0,00	-	0,00	0,00
Valgrind	1,00	0,00	-	0,00	0,00
Skype	1,00	0,00	-	0,00	0,00
Google Software	1,00	0,00	-	0,00	0,00
Vim	1,00	0,00	-	0,00	0,00
Clúster CTE-Power	1,00	2.000.000	4,00	0,40	20,00
Desenvolupador	1,00	10,00	-	-	3.000,00
Avaluació final del rendiment	**20 hores**				**206,80**
Ordinador	1,00	1.900,00	4,00	0,24	4,80
Extrae	1,00	0,00	-	0,00	0,00
Paraver	1,00	0,00	-	0,00	0,00
Google Software	1,00	0,00	-	0,00	0,00
Vim	1,00	0,00	-	0,00	0,00
Clúster CTE-Power	1,00	2.000.000	4,00	0,40	2,00
Desenvolupador	1,00	10,00	-	-	200,00
Documentació i Tancament del Projecte	**65 hores**				**665,00**

	Unitats (Producte o hores)	Preu unitat (€)	Vida útil (anys)	Amortització (€/h)	Preu (€)
Ordinador	1,00	1.900,00	4,00	0,24	15,60
Google Software	1,00	0,00	-	0,00	0,00
Skype	1,00	0,00	-	0,00	0,00
TeXstudio	1,00	0,00	-	0,00	0,00
Vim	1,00	0,00	-	0,00	0,00
Desenvolupador	1,00	10,00	-	-	650,00
Costos Indirectes					**2.300,00**
Despeses d'oficina	5,00	60,00	-	0,00	300,00
Salaris indirectes	100,00	20,00	-	0,00	2.000,00
Total acumulat					**7.465,28**
Contingència	**15% Total acumulat**				**1.119,79**
Total sense IVA					**8.585,07**
Total amb IVA 21%					**10.387,94**

Taula 5.3: Pressupost del projecte

5.8 Control de Gestió

Aquest projecte, com molts en l'àmbit de la informàtica, és candidat a sofrir desviacions en la planificació i, conseqüentment, en el pressupost. En la Secció 2.2 es comenten algunes situacions que poden succeir i enrederir el projecte. Per a controlar aquestes situacions, com s'indica a la Secció 2.3, es duran a terme reunions periòdiques (setmanals i quinzenals). D'aquesta manera, es podran detectar les desviacions i actuar ràpidament per a corregir-les.

Per a solucionar els possibles errors de planificació hem destinat dues partides en el pressupost, com ja s'ha indicat, una de contingència general i una per imprevistos en la tasca *Procés iteratiu de millores*. Serà molt important fer una valoració de cada tasca, avaluant si s'han produït desviacions, quantificant el preu extra o estalvi i les causes. D'aquesta manera es podrà aprendre dels errors comesos per a futurs projectes.

5.9 Desviacions Respecte a la Planificació Inicial

En el moment d'escriure aquestes línies, el projecte es troba ja en la seva recta final i és el moment ideal per fer una valoració sobre la planificació que s'ha mostrat en la Secció 5.2, elaborada durant la fase Gestió del Projecte (curs GEP). Segons la planificació original, actualment, a finals del mes de maig, el projecte hauria d'estar en la recta final de la tasca *Procés Iteratiu de Millores*, però com es pot entendre en llegir la primera frase, actualment

s'està acabant l'última tasca: *Documentació i Tancament del Projecte*. Això suposa, aproximadament, que anem un mes per davant de la planificació.

Hi ha diversos motius que expliquen aquest fet. Primer de tot, en elaborar la planificació inicial es va tenir en compte que totes les hores destinades al projecte estarien integrades dins la jornada laboral, cosa que no s'ha acabat complint. La tasca *Gestió del Projecte* es va desenvolupar pràcticament fora de les hores de treball en la seva totalitat i això va permetre iniciar abans les altres tasques.

També, cal destacar dos successos que han contribuït en el desajust d'aquesta planificació:

- Es va donar la situació explicada en la Secció 2.2 durant un temps bastant ampli de temps, cosa que va dificultar molt executar qualsevol cosa en el CTE-Power. Com s'explica allà, es va aprofitar el temps per elaborar documentació i seguir treballant amb el codi sense mesurar el seu rendiment, però si verificant el seu resultat.

- Fruit del treball realitzat en aquest projecte va sorgir la possibilitat de presentar la feina feta a un *workshop*, així que es van dedicar dues setmanes a preparar el *paper* a enviar. Això va ajudar a avançar la tasca *Avaluació final del rendiment* i redactar una part important de la documentació.

Així doncs, aquests dos *imprevistos* no han tingut un impacte negatiu en la planificació del projecte, més aviat han contribuït, sobretot el segon punt, a accelerar el treball. Gràcies a tots aquests fets, ha estat possible reduir el temps de les tasques *Procés Iteratiu de Millores* i *Avaluació Final del Rendiment*. El temps extra del que s'ha disposat s'ha invertit en l'elaboració de la memòria.

5.9.1 Desviacions en el Pressupost

Com s'ha explicat en la secció anterior, hi ha hagut alguns canvis en la planificació i aquests han tingut algun impacte (o no) en el pressupost final. A continuació s'indiquen quines files de la Taula 5.3 es modifiquen:

- A causa del bon ritme del projecte no es va produir cap imprevist que afectès negativament a la planificació durant la tasca *Procés Iteratiu de Millores*. Això comporta que la partida destinada a imprevistos no ha estat necessària.

- La durada final de la tasca *Avaluació final del rendiment* es va reduir de 20 hores a 12 hores, ja que es va realitzar per preparar el *paper* esmentat a la secció anterior i es va comptar amb l'ajuda dels directors. Aquest fet comporta un estalvi de 81,92 € (es redueixen l'ús de l'ordinador i el salari del desenvolupador), amb un cost final de la tasca de 124,88 €.

- La durada final de la tasca *Procés iteratiu de millores* ha estat d'unes 270 hores, temps que s'ha dedicat a la tasca *Documentació i tancament del projecte*. Aquest fet fa variar el cost de cada tasca individualment, però no canvia el total final.

- La partida destinada a contingències estava pensada per si es produïa algun desajust molt gran, cosa que al final no ha passat. Així doncs, es pot reduir d'un 15 % a un 5 %, passant de gastar 1.119,79 € a 362,98 €.

Per tots aquests motius indicats, el pressupost definitiu quedaria de la següent manera, indicant només el preu per tasca per simplicitat:

	Preu (€)
Gestió del Projecte	768,00
Estudi del codi i Anàlisi inicial	309,20
Procés iteratiu de millores	2.784,80
Avaluació final del rendiment	124,88
Documentació i Tancament del projecte	972,80
Costos Directes	4959,68
Costos Indirectes	2.300,00
Total acumulat	7259,68
Contingència (5% Total acumulat)	362,98
Total sense IVA	7.622,66
Total amb IVA 21%	9.223,42

Taula 5.4: Pressupost Final.

Capítol 6

Sostenibilitat

6.1 Dimensió Econòmica

- **Has quantificat el cost (recursos humans i materials) de la realització del projecte? Quines decisions has pres per reduir el seu cost? Has quantificat aquest estalvi?**
  Sí, s'ha quantificat aquest cost. Es pot trobar a les Taules 5.3 i 5.4.
  Per reduir el cost, s'ha fet servir en tot moment programari lliure, totalment gratuït. Avui en dia, l'oferta de programari lliure és molt àmplia i cobreix totes les necessitats que requereix aquest projecte, així que no ha estat necessari fer ús de cap *software* de pagament.
  Pel que fa a l'estalvi que això significa, no està quantificat, ja que per fer-ho primer s'hauria de trobar un software de pagament que substituís al que hem fet servir, cosa que no s'ha fet.

- **S'ha ajustat el cost previst inicialment al final? Has justificat les diferències?**
  No s'ha ajustat. Hi ha hagut un estalvi de 1.164,52 € respecte a la previsió inicial. Aquesta diferència s'explica a la Secció 5.9.

- **Quin cost estimes que tindrà el projecte durant la seva vida útil? Es podria reduir aquest cost per fer-lo més viable?**
  El projecte no té un cost associat a la seva vida útil. No s'ha de mantenir cap servei ni seguir pagant a ningú un cop acabat aquest treball. Òbviament, es tracta d'un Software i hi haurà gent que el faci servir, però aquest cost no corre en cap cas a càrrec d'aquest projecte.

- **S'ha tingut en compte el cost dels ajustos/actualitzacions/reparacions durant la vida útil del projecte?**
  Si bé és cert que el producte pot tenir algun *bug* que obligui a realitzar ajustos, s'ha provat suficientment i creiem que la probabilitat és molt baixa. D'altra banda, es pot seguir actualitzant el codi per millorar-lo, ja s'ha vist anteriorment que encara hi ha cert potencial de millora, però de fer-se no seria dins el marc d'aquest projecte. Per aquestes dues raons, no s'han tingut en compte els costos associats a aquestes causes.

- **Podrien produir-se escenaris que perjudiquessin la viabilitat del projecte?**
  No hi ha cap escenari que perjudiqui la viabilitat del projecte en l'àmbit econòmic. No es busca treure cap rendiment ni tampoc suposa cap cost afegit durant la seva vida útil.

6.2 Dimensió Ambiental

- **Has quantificat l'impacte ambiental de la realització del projecte? Quines mesures has pres per reduir-ne l'impacte? Has quantificat aquesta reducció?**
  No s'ha quantificat l'impacte ambiental del projecte, però aquest està associat al consum d'electricitat durant el transcurs del projecte i al reciclatge del material informàtic fet servir, tot i que aquest cost no s'ha d'imputar únicament a aquest projecte. Per reduir l'impacte s'han fet servir equips compartits, que es poden reutilitzar per altres feines un cop finalitzat el projecte. L'ordinador portàtil i, com és lògic, el clúster CTE-Power entrarien dins d'aquesta categoria.

- **Si fessis de nou el projecte, podries realitzar-lo amb menys recursos?**
  Començant amb els coneixements adquirits, sí, podria reduir el nombre d'hores dedicades i de computació, reduint el consum elèctric. D'altra banda, crec que he mantingut el consum de recursos al mínim de les meves possibilitats, així que no crec que l'hagués pogut realitzar amb menys recursos.

- **Quins recursos estimes que es faran servir durant la vida útil del projecte? Quin serà l'impacte ambiental d'aquests recursos?**
  Durant la vida del programa, aquest s'executarà a diversos clústers, amb un impacte ambiental en forma de consum elèctric.

- **El projecte permetrà reduir l'ús d'altres recursos? Globalment, el projecte millorarà o empitjorarà la petjada ecològica?**
  Sí, permetrà reduir el consum elèctric. De fet, aquest és un objectiu que s'assoleix indirectament en reduir el temps d'execució del programa, objectiu principal d'aquest projecte. Per tant, amb aquest projecte s'està contribuint a reduir la petjada ecològica, globalment.

- **Podrien produir-se escenaris que fessin augmentar la petjada ecològica del projecte?**
  Aquest projecte no genera cap residu durant la seva vida útil ni quan aquesta s'acaba, ja que es tracta de *software*. Per tant, no es considerà cap escenari on això pugui succeir.

6.3 Dimensió Social

- **La realització d'aquest projecte ha implicat reflexions significatives en l'àmbit personal, professional o ètic de les persones que hi han intervingut?**
  Sí, sobretot a escala professional i personal. Ha estat el projecte més gran (amb més feina associada) en el qual he estat implicat fins ara, i a més a més era jo el principal responsable; la pressió sempre t'acompanya en aquestes situacions, així que tirar endavant és una recompensa molt satisfactòria.

- **Qui es beneficiarà de l'ús del projecte? Hi ha algun col·lectiu que pugui veure's perjudicat pel projecte? En quina mesura?**
  Els principals beneficiaris seran els desenvolupadors de l'aplicació original (ORNL) i els científics interessats a realitzar les simulacions físiques que ofereix. A més a més, qualsevol desenvolupador d'aplicacions paral·leles es pot beneficiar de la feina

que s'ha fet, aprofitant les estratègies de paral·lelització explicades.
No es considera que aquest projecte pugui perjudicar a cap col·lectiu.

- **En quina mesura soluciona el projecte el problema plantejat inicialment?**
  Encara que el programa segueix tenint marge de millora, considero que el projecte ha resolt de manera eficaç el problema plantejat, ja que s'ha aconseguit una versió paral·lela centenars de vegades més ràpida que l'original i a la vegada el consum de memòria també ha baixat uns 90 MB.

- **Podrien produir-se escenaris que fessin que el projecte fos perjudicial per a algun segment particular de la població?**
  El programa en si no està pensat per tenir un impacte en l'àmbit social, sinó que es tracta d'una aplicació per usar-se en l'àmbit científic. A més a més, es mostren les parts del codi rellevants i l'aplicació en si està disponible de manera *online*, al tractar-se d'una aplicació *open source*.

- **Podria crear el projecte algun tipus de dependència que deixés als usuaris en posició de debilitat?**
  El programa no està pensat per generar cap addicció, així que no sembla versemblant que es doni cap situació de dependència envers l'aplicació final.

Capítol 7

Conclusions

En aquest projecte hem pogut veure tot el procés de millorar d'una aplicació orientada a HPC. Concretament, hem pogut veure tots els passos necessaris per analitzar un codi amb detall i millorar-ne el rendiment, així com altres aspectes més associats a la gestió d'un projecte, com per exemple l'elaboració de la planificació i el càlcul d'un pressupost bàsic. Finalment, com ja s'ha introduït en la Secció 5.9, he pogut veure el procés relacionat amb la publicació d'un article, que finalment ha estat acceptat i es troba a l'espera de ser publicat [11].

Centrant-nos en el treball realitzant sobre DMRG++, cal destacar la contribució cap als científics que utilitzen regularment l'aplicació. Mitjançant el sistema de tasques amb dependències, hem aconseguit arribar a un speedup de 20,54× respecte la primera versió (*Primera Tasquificació*) presentada sense OpenMP, o de 891× en comparació amb la paral·lelització original del codi. A més a més, aquest guany s'ha aconseguit reduint el consum de memòria en 90 MB, cosa que incrementa el seu valor.

Tots aquests canvis s'han fet tenint present la producció d'un codi net i llegible, tenint en compte que el consumidor final no som nosaltres mateixos, sinó la gent responsable a l'ORNL. Per això mateix, s'ha intentat sempre que els canvis en el codi fossin mínims, mantenint la seva claredat, i fins i tot s'ha arribat a reduir el nombre de pragmes d'OpenMP respcte a la paral·lelització original.

També, voldria remarcar la utilitat que pot tenir el treball per altres usuaris que facin ús d'OpenMP. Altres persones treballant amb *kernels* que presentin desbalancejos com els que s'han vist en aquest programa es poden veure beneficiats de les pràctiques que es mostren en aquest treball, adoptant-les per a les seves aplicacions.

Com a feina futura hi ha diversos camins a emprendre, que podrien servir com a punt de partida de nous projectes. Es pot seguir treballant en la direcció vista aquí, arreglant el desbalanceig de la versió *Tasques Aniuades*. També, seria interessant explorar algunes de les noves possibilitats que ofereix la versió 5.0 d'OpenMP, com ara el tipus de dependència *mutexinoutset*. Finalment, una altra opció és desenvolupar un codi híbrid OpenMP i MPI, que podria ajudar a solucionar l'efecte NUMA que s'ha vist en passar de 20 a 40 cores.

Per acabar, el projecte m'ha servit a títol personal com a punt d'entrada al món de la recerca i al laboral, d'una manera més seriosa. Trobo que els TFGs són una gran oportunitat per iniciar-se dins la investigació, sigui de la modalitat que sigui, un punt que es deixa

bastant de banda durant la carrera i que s'hauria d'incentivar més. Gràcies a aquest treball, he pogut arribar a publicar un article (com ja s'ha dit) i viure el procés de presentació en un *workshop*. A més a més, també ha sorgit l'oportunitat de fer un *internship* a l'ORNL, fruit de la col·laboració en aquesta feina, experiències que espero que siguin molt enriquidores.

Bibliografia

[1] DMRG++ miniapp repository. https://github.com/arghyac007/DMRG-openMP - Last accessed June 2019

[2] Paraver website. https://tools.bsc.es/paraver - Last accessed June 2019

[3] POP-COE, Performance Optimisation and Performance Center of Excellence in Computing Applications. https://pop-coe.eu - Last accessed June 2019

[4] Power9 CTE User's Guide. https://www.bsc.es/support/POWER_CTE-ug.pdf - Last accessed June 2019

[5] Top500 List. https://www.top500.org/lists/2019/06/ - Last accessed June 2019

[6] Trello, Projecte Management and Organization Tool. https://trello.com/about - Last accessed June 2019

[7] Valgrind, System for debugging and profiling Linux programs. http://valgrind.org - Last accessed June 2019

[8] Alvarez, G.: The density matrix renormalization group for strongly correlated electron systems: A generic implementation. Computer Physics Communications **180**(9), 1572–1578 (2009)

[9] Cajas, J., Houzeaux, G., Vázquez, M., García, M., Casoni, E., Calmet, H., Artigues, A., Borrell, R., Lehmkuhl, O., Pastrana, D., Yáñez, D., Pons, R., Martorell, J.: Fluid-Structure Interaction Based on HPC Multicode Coupling. SIAM Journal on Scientific Computing **40**(6), C677–C703 (2018). https://doi.org/10.1137/17M1138868, https://doi.org/10.1137/17M1138868

[10] Chatterjee, A., Alvarez, G., D'Azevedo, E., Elwasif, W., Hernandez, O., Sarkar, V.: Porting DMRG++ Scientific Application to OpenPOWER. In: International Conference on High Performance Computing. pp. 418–431. Springer (2018)

[11] Criado, J., Garcia-Gasulla, M., Sirvent, R., Labarta, J., Chatterjee, A., Hernandez, O., Alvarez, G.: Optimization of Condensed Matter Physics Application with OpenMP Tasking Model. In: International Workshop on OpenMP. pp. In–press.

[12] Duran, A., Ayguadé, E., Badia, R.M., Labarta, J., Martinell, L., Martorell, X., Planas, J.: OmpSs: a proposal for programming heterogeneous multi-core architectures. Parallel Processing Letters **21**(02), 173–193 (2011)

[13] Elwasif, W., D'azevedo, E., Chatterjee, A., Alvarez, G., Hernandez, O., Sarkar, V.: MiniApp for Density Matrix Renormalization Group Hamiltonian Application Kernel. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER). pp. 590–597. IEEE (2018)

[14] Garcia, M., Labarta, J., Corbalan, J.: Hints to improve automatic load balancing with LeWI for hybrid applications. Journal of Parallel and Distributed Computing **74**(9), 2781 – 2794 (2014). https://doi.org/http://dx.doi.org/10.1016/j.jpdc.2014.05.004, http://www.sciencedirect.com/science/article/pii/S0743731514000926

[15] Garcia-Gasulla, M., Mantovani, F., Josep-Fabrego, M., Eguzkitza, B., Houzeaux, G.: Runtime mechanisms to survive new hpc architectures: A use case in human respiratory simulations. The International Journal of High Performance Computing Applications p. 1094342019842919 (2019)

[16] Gautier, T., Pérez, C., Richard, J.: On the Impact of OpenMP Task Granularity. In: International Workshop on OpenMP. pp. 205–221. Springer (2018)

[17] GitHub: The world's leading software development platform github. https://github.com - Last accessed June 2019

[18] Grinberg, L., Bertolli, C., Haque, R.: Hands on with OpenMP4. 5 and unified memory: developing applications for IBM's hybrid CPU+ GPU systems (Part I). In: International Workshop on OpenMP. pp. 3–16. Springer (2017)

[19] Llort, G., Servat, H., González, J., Giménez, J., Labarta, J.: On the Usefulness of Object Tracking Techniques in Performance Analysis. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 29:1–29:11. SC '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2503210.2503267, http://doi.acm.org/10.1145/2503210.2503267

[20] Martineau, M., McIntosh-Smith, S.: The productivity, portability and performance of OpenMP 4.5 for scientific applications targeting Intel CPUs, IBM CPUs, and NVIDIA GPUs. In: International Workshop on OpenMP. pp. 185–200. Springer (2017)

[21] Meng, Z., Koniges, A., He, Y.H., Williams, S., Kurth, T., Cook, B., Deslippe, J., Bertozzi, A.L.: OpenMP parallelization and optimization of graph-based machine learning algorithms. In: International Workshop on OpenMP. pp. 17–31. Springer (2016)

[22] Message Passing Interface Forum: Message Passing Interface Standard Version 3.1. https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf - Last accessed June 2019

[23] Moore, G.: Llei de Moore. https://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html - Last accessed June 2019

[24] OpenMP Architecture Review Board: OpenMP 4.5 Specification. https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf - Last accessed June 2019

[25] Rising, L., Janoff, N.S.: The Scrum software development process for small teams. IEEE software **17**(4), 26–32 (2000)

[26] Rupp, K.: 40 Years of Microprocessor Trend Data. https://www.karlrupp.net/wp-content/uploads/2015/06/ - Last accessed June 2019

[27] Sadasivam, S.K., Thompto, B.W., Kalla, R., Starke, W.J.: IBM Power9 processor architecture. IEEE Micro **37**(2), 40–51 (2017)

[28] Schuchart, J., Nachtmann, M., Gracia, J.: Patterns for OpenMP task data dependency overhead measurements. In: International Workshop on OpenMP. pp. 156–168. Springer (2017)

[29] Teruel, X.: OmpSs Quick Overview. https://nanohub.org/resources/19155/download/ompss-01-quick_overview.pdf - Last accessed June 2019

[30] White, S.R.: Density matrix formulation for quantum renormalization groups. Physical review letters **69**(19), 2863 (1992)